Canada $4.25
USA $3.50

FOR THE Amiga Transactor

MARCH 1989: VOLUME 1, ISSUE 6

THE MAGAZINE FOR AMIGA PROGRAMMERS

- Three-dimensional fractals *by Cathryn and Eric Graham*
- Assembly language programming *a new series by Jim Butterfield*
- System memory allocation and its pitfalls *by Eric Giguere*
- DNet: a two-Amiga network; theory and application *by Matthew Dillon*
- Mastering the Raster *a new column by Derek Grime*
- Hard disk directory management *by Dan Schein*
- Prop gadgets with embedded numeric display *by Hobie Orriss*
- Implementing an ARexx interface *by Ben Blish*
- Hard Copy *a new column on documentation by Tim Grantham*
- **Plus** regular columns by Larry Phillips, Don Curtis and Steve Ahlstrom

Bathers (after Cézanne) by Christine Delezenne

0 29074 12122 03

WIN A MONITERM!

Build an expert system application with the MAGELLAN™ artificial intelligence expert system building tool and win a MONITERM Viking I high resolution monitor for your AMIGA.
Imagine the possibilities: the power of the AMIGA coupled with the flexibility of MAGELLAN!

Categories for the First Annual MAGELLAN Application Contest are:

Most Productive Application
Business or Diagnostic systems, decision assistance tools for financial investment, production control systems, or inventory management systems.

Most Creative Application.
Music or Animation systems, or innovative approaches to traditional expert systems.

Most Bizarre Application.
Let your imagination run wild!

3 Grand Prize Winners will receive:
A Moniterm Monitor
A MAGELLAN Developer Package
Retail Value $3000.

2 Runners-Up in each category will receive:
MAGELLAN Developer Package
Retail Value $1000.

All finalists will also receive a MAGELLAN designer sweatshirt.

All entries must be received by Emerald Intelligence by June 1, 1989. Contest void where prohibited. Contest Winners will be announced at Ami-Expo Chicago in July 1989. All entries become the exclusive property of Emerald Intelligence Inc., and cannot be returned. Winning entries may be used in a promotional campaign and any entry may be included in promotional materials or application manuals. For a complete set of rules and an entry form, send a self addressed stamped envelope to MAGELLAN Contest Rules, c/o Emerald Intelligence.

MAGELLAN™

THE EXPERT SYSTEM SOFTWARE

Emerald Intelligence

3915-A1 Research Park Drive, Ann Arbor, Michigan, U.S.A. 48108. (313) 663-8757.

MAGELLAN is a trademark of Emerald Intelligence Inc. AMIGA is a registered trademark of Commodore-Amiga Inc. Moniterm is a trademark of Moniterm Corporation.

The BEST AMIGA DISKS

Quality User Supported Software works with all Amiga systems

$7 Each Buy 1-4 Disks

$6 Each Buy 5-14 Disks

$5 Each Buy 15 or more Disks

So easy to Use

- ▼ No Computer Experience Necessary
- ▼ Instructions for each Order
- ▼ FREE Phone Support

So Easy to Order

- ▼ FREE Membership ▼ FREE 800-# for Orders
- ▼ FREE Same-day Shipping
- ▼ UPS 2nd Day Air Service when you need it Now!
- ▼ FREE Catalog ▼ Your Satisfaction Guaranteed

THE TOP 10

☐ **#131 PacMan '87** - Great sound and graphics. Adds new elements to PacMan. Saves Top 10.
☐ **#127 Wheel of Fortune** - A great computer version for multiple players. It even talks.
☐ **#23 Monopoly** - Enjoy great graphics and sound while playing three tough computer opponents.
☐ **#37 Business Programs** - Included are an address book, an amortization program, a talking mail manager, and a label printer.
☐ **#27 Amoeba Invaders** - A better Space Invaders!
☐ **#140 Virus Killer** - Everyone needs this! Makes it easy to detect and eliminate known viruses.
☐ **#115 Word Processor** - Lots of features.
☐ **#134 Applications** - Label maker/printer, grocery list maker, and AMIGazer - a star viewing program.
☐ **#142 Q-Bert** - Like the popular arcade version.
☐ **#139 Bull Run** - Great Civil War strategy board game, with impressive graphics and sound.

1-800-444-5457 for Orders

THE BEST OF THE REST

503-664-5953 - Foreign Orders

BUSINESS/HOME

☐ **#116 Business II** - VC-Spreadsheet, HP-10c calculator, and several diversions for when the boss isn't looking!
☐ **#117 Business III** - DataBase, a bunch of great new fonts, RSLClock-great clock utility, AmigaSpell.

LANGUAGES

☐ **#9 FORTH** - Two versions of the FORTH programming language.
☐ **#50 XLISP** - A version of LISP, the artificial intelligence language. Includes documentation.

COMMUNICATIONS

☐ **#4 Communications** - Starterm and Aterm are both included. Both run from Workbench or CLI and are comparable to communications packages selling for $50+. Full control of baud rates, phone directory, all protocols, auto chop of files, and many other extras. Works well with the Avatex 1200 modem and others. Source code included.
☐ **#90 Modem Madness!** - Terminals include StarTerm, ATerm, and Kermit. Telecommunication utilities such as Archive are also here. Other types of programs and utilities are also included in the price of admission.

UTILITIES/APPLICATIONS

☐ **#60 AmigaBasic** - Two programs that are truly of commercial quality, Cell-Animate and Graphit. Some Deluxe Paint picture files are also included.
☐ **#97 Tutorial Disk1** - A disk full of information and programs to instruct Amiga Programmers and users. Several C & ASM source files are included.
☐ **#98 Tutorial Disk2** - More of the best of Amiga Information.
☐ **#101 Utilities** - Many new utilities like Timeset - a time setting utility and DirCopy-a great copier (very quick), and about a dozen more. Some new fonts are also included on this disk.
☐ **#105 Potpourri I** - This disk contains several different kinds of programs, some of the highlights are: PopCLI2-evokes a new CLI window at the press of a button; PSound-sample sound recorder and editor; 3-D Breakout; DiskCat-catalogs and organizes disk files; IconMaker-makes icons for most programs so that they can run from Workbench; FKey-template maker.
☐ **#129 Amiga Utilities II** - A hard disk backup; Target-sounds a gunshot whenever the left mouse button is pressed; Dpaint Tutor; WinSize-change window size from CLI easily, and lots more.
☐ **#132 Videomaker Utilities** - This disk is packed with utilities to make your desktop videos easier to produce and more professional looking.
☐ **#133 DOS Helper** - A program designed to help you with the AmigaDOS commands. Can be activated from icon of the CLI. Supports multitasking, so that you can refer to it when you need it. As usual, there are other good programs included on the disk.
☐ **#135 Applications II** - Long Movie-plays several IFF pictures in fast succession, creating animation. QuickBase-a mail manager DBase. Persmait-a DataBase for keeping records of friends, family, associates, customers or employees. MORE.
☐ **#146 Calendar** - A very good personal calender for birthdays, holidays, meetings, bills and other events. Excellent graphics. Calendar program also has a diary. Other programs include some graphics and Checkbook.
☐ **#150 Textcraft+ Demo** - A demo of a very good word-processing program of high quality. Menu driven. Has a lot of helps to show you how. Textcraft+ also has a Speller Check available.

SOUND/MUSIC

☐ **#18 Future Sound Demo** - Another great sound demo of digitized sound. Includes the wicked witch of OZ, breaking dishes, sea gulls, car crash, ducks, others.
☐ **#30 Super Sounds** - Great digitized sounds from movies like Star Trek, 2001, James Bond movies, Star Wars, and Starman, Is it real or is it the Amiga?!!!
☐ **#77 Instruments** - Turn your keyboard into 25 different musical instruments.

SLIDE SHOWS

☐ **#1 Norman Rockwell** - 17 beautiful digitized Rockwell paintings in this self-running slideshow presentation. You've got to see these!
☐ **#55 EA Demo and Polyscope** - More great graphics for your enjoyment.
☐ **#67 DPSlide8** - Over 30 slides of all sorts on this self-running slideshow.
☐ **#94 Diga-View** - This one shows the digitizing process in stages. Several good pictures are included. Other programs also included.
☐ **#95 DigaSlide11** - Another in the great series of slideshows with great artwork. Self-running with over 25 pictures.
☐ **#108B Juggler** - Famous demo that shows the beautiful graphics of the Amiga and just how powerful this program is. It is easier to run than 108A, but only has the one demo on it.

GAMES

☐ **#38 Basic GrabBag2** - Around 25 programs of various types. Many of these are must-haves. At less than $.20 each, you can't go wrong!
☐ **#44 Games3** - More great games including Life, Vegas Slot Machine, Reversi, others.
☐ **#52 Basic Games** - Tons of ABasic games - discover some treasures!
☐ **#61 ABasic GrabBag1** - Only about 100 of all types!!!!!
☐ **#102 Sinking Island** - Return to Sinking Island is an excellent adventure game. Well worth the price - hours of enjoyment!!!
☐ **#114 Potpourri X** - Othello, A key-shortcut program for AmigaDos. Various new tools, automatic printer-driver generator, much more.
☐ **#118 Great Graphic Games** - Includes Missle Command, 2-D Triclops, Cosmo-asteroids clone, BrakeOut, Yatzee, Hack and more.
☐ **#121 Backgammon** - A great game from David Addison.
☐ **#122 Solitaire** - Two versions by David Addison.
☐ **#123 Cribbage** - Take on the computer or a friend.
☐ **#124 Milestone** - A great computer version of Miles Bournes by the author of Monopoly for the Amiga, David Addison.
☐ **#125 Othello** - A great 3-D version of this popular game.
☐ **#128 Space Games** - Cosmoroids (like asteroids) and Gravity Wars highlight this disk just full of games.
☐ **#137 Blackjack** - A full-featured game which allows pair-splitting, double-down, etc. Bandit-play the slot machines without going to Vegas!!! More.
☐ **#141 Dominoes** - Dominoes game with great graphics. Also Tic Tac Toe, Drawing and Molecules programs.
☐ **#147 Jackland/Graphics** - Adventure clue game. Also some great pictures (graphics), a useful utility called Quickbase, and a fun program called Thingies which you will enjoy!!!
☐ **#148 Boulder Dash** - Very popular game with excellent graphics and has several challenging levels. This disk is full- It has Othello, Life3 and many useful utilities.
☐ **#151 4 in a Row** - A fun, but challenging game you play against the computer. There is an excellent Demo "MandFXP-D3", a utility or 2 and the fun TARGET - A weirdo thingy.

MISCELLANEOUS

☐ **#88 Amiga Basic Programs** - Over 50. Games, utilities, applications, entertainment, and finance. Also included is a program that allows you to use IFF files in your Amiga Basic programs.
☐ **#119 mCAD** - A full-featured computer-aided design program.
☐ **#136 Graphics2** - Border Set-useful for desktop publishing and video, making cards, coupons or menus, and your own artwork. Xicon-allows you to run AmigaDOS commands or programs from Icon.
☐ **#144 Christmas Animations** - 10 beautiful scenes and graphics with sound. Great scenes include: Lit Candle, Elves, The Christmas Tree, The Manger, Season's Greetings, Holly Wreath, Chimney Smoke, Church Bells, and Walking Home.
☐ **#145 Animations 3** - More Great Animations. 3 very good demos plus 3 workbench pictures and Blobs.

Name _____
Address _____
City _____ State ____ Zip ____
Phone (____) _____
Visa/MC ____-____-____-____
Sig. _____
Exp. Date _____ ☐ Send Catalog

Disks Ordered ____ x ____ $ ____
☐ SHIPPING Free U.S. $ ____
Canada - 25¢ per disk. Foreign - 50¢ per disk
☐ COD $ ____
(add $4 if you require COD)
☐ U.P.S. 2nd Day Air (Add $3) $ ____
TOTAL $ ____
☐ Check/MO ☐ Visa/MC ☐ COD

Software Excitement!

P.O. Box 5069 • Central Point, OR 97502

VISA ORDER TODAY MasterCard

1-800-444-5457

Volume 1, Issue 6

Publisher
Antony Jacobson

Vice-President Operations
Jeannie Lawrence

Vice-President Advertising
Maximillian Jacobson

Assistant Advertising Manager
Julie Cale

Editors
Nick Sullivan
Chris Zamara

Assistant Editor
Malcolm O'Brien

Contributing Writers
Steve Ahlstrom
Charles B. Blish
Anthony Bryant
Jim Butterfield
Betty Clay
Don Curtis
Matt Dillon
John Faichney
Andy Finkel
Brad Fowles
Eric Giguère
Cathryn Graham
Chris Gray
Derek Grime
Jay Gross
Louis Markoya
Eugene P. Mortimore
Bryce Nesbitt
Hobie Orris
Rob Peck
Larry Phillips
Dan Schein
Steve Simpson
John Toebes
Victor A. Wagner
Clark Williams
James D. Williams
Doug Winger
David Wood

Cover Artist
Christine Delezenne

for the AMIGA Transactor
The Magazine for Amiga Programmers

Implementing an ARexx Interface 24
by Charles B. Blish
Design decisions involved in adding ARexx support to an application.

Memory Allocation On The Amiga 28
by Eric Giguère
Don't take memory allocation and freeing for granted - fragmentation can cause you trouble!

Breakpoint, Part 4 32
byVictor A. Wagner
A comparison of the two source debuggers: Manx's *sdb*, and Lattice's *CodePRobe*.

Prop-Gadgets With Embedded Numeric Display 38
by Hobie Orris
Display numeric values where they're most convenient - on the knob of the 'slider' gadget!

Three Dimensional Fractals 42
by Cathryn and Eric Graham
Fractals have provided Amiga users with breathtaking two-dimensional displays. In this article, Eric and Cathryn apply their solid modelling methods to produce 'snowflakes' and realistic trees.

DNet and Sliding Window Protocols 52
by Matthew Dillon
An in-depth technical explanation of network protocols and Matt's *DNet* two-computer link.

The *TransAmi* DNet Network 60
by Chris Zamara
An overview of a practical and simple-to-implement network based on *DNet*.

Whap! - A Beta tester's experiences 62
by Doug Winger
The saga of the test/development cycle of one program, from first version to release

Getting Mass Storage Under Control 66
by Dan Schein
An explanation of AmigaDOS directories from the ground up; manage your storage efficiently.

Scripter - A text editing system for writing scripts 70
by Jay Gross
Your Amiga lets you paint, sculpt, and write great novels, but try writing a simple script! With these short AmigaBasic programs - and good old *Ed* - it becomes less of a chore.

Departments and Columns

ViewPort 4

by Larry Phillips

Does a product fall short of the mark because of its capabilities, or the way we evaluate it?

ACCESS - The best of non-commercial software 6

by Steve Ahlstrom

A look at a collection of utilities by Khalid Aldoseri that are making life easier for Steve.

Amiga Dispatches 12

by Don Curtis

Problems with installing the Fast File System on a *JANUS* drive, and comments about the A500 and the current state of Amiga software.

Mastering The Raster 16

by Derek Grime

The first instalment of a new column that looks at using the Amiga from a professional graphic artist's perspective. In this article, Derek points out some glaring gaps in the software arena.

Assembly Language Programming 18

by Jim Butterfield

Another new column, by Jim Butterfield, the master of the tutorial. In this first instalment, a self-contained 'Hello, world!' program is used to illustrate some fundamental concepts.

Hard Copy - books, documentation and information 76

by Tim Grantham

Yet another new column explores 'paperware': books, manuals, and information sources of all kinds. This time around, Tim reviews two excellent books: *Understanding PostScript Programming*, and *COMPUTE!'s Amiga Machine Language Programming Guide*.

About the cover: *Bathers (after Cézanne)* by Christine Delezenne

Christine Delezenne is a French artist visiting Canada. After graduating in Philosophy of Art at the University of Lyon III, Christine studied at an experimental art school called Emile Cohl, where the Amiga was used in a course on computer art. Using Deluxe Paint II to form the structure of the 'painting', and Digi-Paint to create the 'water-colour' effects, Christine did a wide variety of classical-style art on the Amiga, including some, like this picture, after the style of well-known artists. Christine's art is a striking example of how non-technical people ("I have trouble using a television set", she jokes) can produce outstanding creations on the Amiga with the right tools.

Transactor for the Amiga is published monthly by Croftward Publishing Inc., 10-85 West Wilmot Street, Richmond Hill, Ontario, L4B 1K7. ISSN# 0838-0163. Canadian Second Class Mail Registration No. 7690, Gateway-Mississauga, Ont. USPS Postmasters: send address changes to: Transactor, PO Box 338, Station C, Buffalo, NY, 14209.

Croftward Publishing Inc. is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names are registered trademarks of Commodore Inc.

Subscriptions: see order card at centre of magazine for complete ordering information and prices.

Send all subscriptions to: Transactor Subscriptions Dept. 10-85 West Wilmot Street, Richmond Hill, Ontario, Canada, L4B 1K7, (416) 764-5273. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to our Richmond Hill HQ. For best results, use the postage paid card at the centre of the magazine.

Quantity Orders: In Canada: Ingram Software Ltd., 141 Adesso Drive, Concord, Ontario, L4K 2W7, (416) 738-1700. In the USA: IPD (International Periodical Distributors), 11760-B Sorrento Valley Road, San Diego, California, 92121, (619) 481-5928; ask for Dave Buescher.

Editorial contributions are always welcome. Only original, previously unpublished material will be considered. Program listings and articles, including BITS submissions, of more than a few lines, should be provided on disk. Preferred format is Amiga 3.5-inch diskettes with ASCII text files. Manuscripts should be typewritten, double-spaced, with special characters or formats clearly marked. Photos should be glossy black and white prints. Illustrations should be on white paper with black ink only. IFF picture files or Aegis Draw-Plus structured graphics files are preferred to hardcopy illustrations when possible.

All material accepted becomes the property of Croftward Publishing Inc., except by special arrangement. All material is copyright by Croftward Publishing Inc. Reproduction in any form without permission is in violation of applicable laws. Write to the Richmond Hill address for a writer's guide.

The opinions expressed in contributed articles are not necessarily those of Croftward Publishing Inc. Although accuracy is a major objective, Croftward Publishing Inc. cannot assume liability for errors in articles or programs. Programs listed in Transactor for the Amiga, and/or appearing on Transactor for the Amiga disks, are copyright by Croftward Publishing Inc. and may not be duplicated or distributed without permission.

Production
In-house with Amiga 2000 and Professional Page
Final output by Vellum Print & Graphic Services, Inc., Toronto

Printing
Printed in Canada by Bowne of Canada Inc.

Viewport

Amiga software and Sturgeon's Law

by Larry Phillips
Copyright © 1989 Larry Phillips

Larry Phillips is an Amiga hardware and software hacker from Vancouver, British Columbia, and a SysOp on CompuServe's Amiga forums.

Someone once told author Theodore Sturgeon, "Ninety percent of science fiction is crap." Sturgeon's answer was to the point. He said, "Ninety percent of *everything* is crap." This little gem has since become known as Sturgeon's Law, and depending on your viewpoint, may be more or less true. The important thing is that it is your viewpoint that determines the truth or falsity of Sturgeon's Law as it pertains to whatever you are applying it to.

I recently had occasion to respond to a comment on a product that is soon to be released for the Amiga. I disagreed, strongly, with the comment. Before I get into the comment or the reasons for my disagreement, here's a little background.

PostScript interpreters are usually built into a printer, such as the NEC LC-890 or the Apple LaserWriter, or into a typesetting machine like a Linotron. PostScript code is sent to the printer, which is run through the interpreter, and produces output at the resolution of the output device, which can range from 300 DPI (Dots Per Inch) for the laser printers, to 2540 DPI for the Linotron. PostScript allows the definition of objects that are independent of the resolution of the device. These objects can be geometric, such as circles, squares, lines, or combinations of them. They can be individual characters in a particular font, in virtually any size you want them to be.

Because of the resolution-independent nature of PostScript, the output of a PostScript-capable device will look as good as it possibly can within the resolution of the device. A 300 DPI laser printer will not produce output that looks as good as that from a 1200 DPI Linotron, and a 2540 DPI Linotron's output will look even better. This resolution-independence is already put to good use by typesetters and publishers. Take a look at the page you are reading right now. Look at the quality of the characters, at the border, at the different font styles. This page was produced with PostScript - all of it, even the borders. It was produced on a Linotron typesetting machine, at a resolution of about 1200 DPI. Now I can tell you that Transactor does not own a Linotron. They are expensive and, more to the point, they are unnecessary for the purpose of setting up a page such as this, so the expense of one would wasted.

When the *TransAmi* editors are setting up a page like this one, they use a PostScript-capable laser printer attached to an Amiga. The 300 DPI resolution is plenty good enough for them to tell what the page is going to look like. They can 'preview' the page for appearance, and know that when it looks right, it will look right when typeset on a more expensive machine. Typesetters are very picky people when it come to quality. They are able to tell almost instantly what resolution was used in the production of a book or magazine. They will tell you that 300 DPI is simply not good enough, and that 1200 DPI is only good for some purposes.

Have you ever looked at the output from a PostScript-capable laser printer? Chances are you thought it was pretty darned good, perhaps even 'great', but only if you are not a typesetter. This is what I meant when I said that it all depends on your point of view. Given sufficient training, you too could see just how useless a 300 DPI printer really is.

Back to the comment. A company called Pixelations is hoping to release a product in February, called PrintScript. PrintScript is a PostScript interpreter, but with a difference. It will do its work in your Amiga's memory, instead of in the memory of an expensive laser printer. Further, it will do its work to the resolution you require for your printer, which means that you can output PostScript to your 9 pin, 24 pin, inkjet, or even a laser printer that cannot normally handle PostScript. Basically, it will take PostScript code and build a bitmap that 'fits' your printer's capabilities, then send it to the printer as a graphic dump. The PostScript code itself can be generated manually, in an editor, or by a program such as Professional Page. It will sell for $89.00 US, and so it is not in the price range you would normally think of as being 'professional' software.

The comment was based on the fact that initially, PrintScript will only come with two fonts. They will look much like Times and Helvetica, two of the most popular fonts available in PostScript printers. The person who made the comment is a

professional, possibly a typesetter, and his comment was that "everyone is wasting their time on a Postscript program that does not support a large collection of standard fonts". From his point of view - the point of view of the professional, that is - he is absolutely right. What of the other point of view though - that of the amateur, the person who wants to produce the occasional newsletter, mailing label, or business letter?

It has always been a source of wonder for me that there are products available that are normally very expensive, but are produced in such a way as to be reasonably priced, with fewer features or lesser quality. Nobody would even think of trying to use a Mattel video camera to produce an episode of a network television program, yet many of these units are sold for children's use. Do they care about the plastic lenses, the relatively poor video quality? No, of course not. They point it at their friends and have them do goofy things, and generally get a lot of fun and their money's worth out of it. There are many examples I could cite, from cheap plastic cameras, to the stereo systems that come with two cassette decks, a built-in turntable, receiver, power amplifier and speakers, all for $59.95.

It is the same with software. PageSetter was an interesting program in that it was aimed at the home market. Some thought it was a poor program because it did not offer professional quality output and features, but I think they were missing the point. Professional Page is the program they wanted, and I haven't heard anyone complain about it, except for saying that it costs too much. Software is no different from hardware in this regard. It costs money to provide features and quality, and you cannot expect to have the best software at low prices. The technique is the same when you are shopping for software and hardware. You decide what you need a package to do, and look for the package that fits the bill, and hopefully the wallet.

PrintScript will fill a need. The need is for the ability to produce the best output of which your printer is capable, and it will do so at a decent price. It will eventually have more fonts available, and will undoubtedly gain features as well. That it is not a professional tool at this time is the reason that a professional lumped it into that ninety percent that Theodore Sturgeon talks about, but to the home or occasional user, it is part of the ten percent that isn't crap. There will still, however, be those that expect more from it, and who will complain that it is a waste of time.

The only problem with the viewpoint argument comes from those products that are targeted to the professional and advertised as such, when in fact they are little more than home market products. I don't care that my CAD program cannot match a professional-quality program, because it suits my limited needs and cost a lot less than a professional-level program. If it had been *presented* as a professional-level program, I would likely have been quite upset, and a buyer who had need of a professional-level program would certainly have been upset, regardless of price. I would think that it would pay a company to consult with professionals and take great pains to describe their product accurately, rather than risk the ire of those who need the capabilities described in such glowing prose on the package.

In conclusion, what can we say about Sturgeon's Law as it applies to the current line up of Amiga software? Well, if you have professional needs, it's pretty close to the mark, if not a little conservative. If you are an average home user, the ninety percent figure is probably high by a factor of at least two.

If you are a software developer, why not take a good, hard look at your product, with the help of professionals in the appropriate field, and call your product what it really is? Don't be afraid to bring out a product that suits the home market, label it as such, and use that as a base for further development.

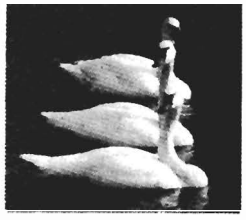
* * * * *

Last month, I promised to supply source code for a serial module as part of a series of modules for ROBBS (REXX Object Building Block System). Unfortunately, I have had to postpone this because of a rather nasty, last minute bug that cropped up and just had to be fixed. I fully expect to have it ready for next month, and ask that you have patience. To paraphrase a well known commercial message, "I will publish no source before its time."

CygnusEd Professional

by CygnusSoft Software. Published by ASDG Inc.

"The New King of Editors,"
68000er, February 1988



"This editor is definitely one of the best I've used on any machine, and I've used many."
The Amiga Sentry, Sept. 1988

CygnusEd Professional brings unparalleled levels of flexibility and speed to the **Amiga.** features include:

Crash Recovery. Variable screen resolution up to 1000 by 800. Full Macro capability. Over 200 functions and variables available through AREXX. Hot Key operation. Screen refresh at over 30,000 cps. Search and replace at over 100,000 cps. Blitter based horizontal and vertical scrolling. Vertical and Horizontal block cut and paste. Many Word Processing features. Many Programmer features. Edit multiple files on screen at same time.

Asynchronous Printer support. Super interprocess communications facilities. DOS and AREXX commands can be launched from inside the editor. User definable "bookmarks". International keyboard and video support. Mouse or keyboard based Turbo Scrolling. Keyboard and mouse can be interchanged for everything except entering text. Will create icons for WorkBench users. Edits binary files. And **MUCH more!**

ASDG INCORPORATED
925 Stewart Street
Madison, WI 53713
(608) 273-6585

Available At Your Local Dealer $99.95 U.S. Dollars

CygnusEd Professional Copyright 1988 By CygnusSoft Software. Amiga and Workbench are trademarks of Commodore-Amiga Inc
Developers are encouraged to contact ASDG for possible publication of your products

ACCESS

The world of non-commercial software

by Steve Ahlstrom
Copyright © 1988 Steven Ahlstrom

As the primary Sysop on CompuServe's Amiga forums, and a veteran Amiga programmer, Steve Ahlstrom sees just about every notable public domain and shareware program that gets written, and in this column he tells you about the ones that have recently caught his eye. If you have some information you'd like to share through this column, you can leave a message to Steve on CompuServe (his PPN is 76703,2006), or send the info to us and we'll pass it on.

Last time I took a look at *Disk Storage* by Khalid Aldoseri. Because it was such a nice, well done and to-the-point utility, I started looking at other programs he's written. It seems he's very prolific, with quite a library of utility programs. I'm going to talk about several of them from his collection.

First, though, a little background info on Khalid (since he is anything but your 'average' Amiga user). He was born, and currently lives, in Manama, Bahrain. At 21 he received his Master's Degree in Management from a Swiss University. Bahrain is a very small Persian Gulf island country (231 square miles with a population of about a quarter million people) located off the coast of Saudi Arabia. Bahrain is a major international banking center. Now 23, Khalid works as a bank inspector. This position requires frequent international travel.

Having started his computing experience on the PET 2001 he moved to the VIC-20, the C64 (on which we wrote an Arabic word processor), and then to the IBM PC. About the time he decided he needed to upgrade from the PC to an AT he read an article in *Byte* Magazine about the Amiga. He immediately cancelled his order for the AT and waited for his Amiga.

Khalid's system, or at least the set-up he calls his 'programming system' is one to make any Amiga owner envious! It consists of an 8MB A2000, CSA 68020/68881 combo, half a megabyte of 32-bit RAM, 2090A, 100MB Conner hard disk, MicroWay flickerFixer, and NEC MultiSync II. He has other Amiga systems that are dedicated for business purposes including a fully functional Amiga 500 and hard disk in an SX-64 case - Khalid is a bit of a hardware hacker too.

Khalid writes mostly small utilities that he finds handy for himself. These utilities are indeed handy for many of us!

Locks

Have you run across a program that puts an AmigaDOS lock on a file and does not release it when the program terminates? What do you do? Well, you can ignore it, but there may be times you would rather release the lock than ignore the problem. You could always reboot. That will release the lock. Again, that may not be an appropriate solution.

Now there's *Locks*. If you want to remove all locks on a device, make sure that the locks have not been put there by an assign. If so, remove the assign before unlocking the device. If you don't the next time you run **assign** you will be met by the turbaned one.

This program is small and to the point, but it is extremely powerful - so much so, if you don't know what you're doing you can really mess up your system. If you understand what AmigaDOS file locks are all about, proceed with caution. If you don't, stay away from this one until you do! For information on file locks see the *AmigaDOS Manual* or just about any book on Amiga programming.

Float

Have you ever used the Snapshot feature of Workbench only to find out that you can't return the icon to its original 'floating' state? *Float* will take any Workbench icon (Disk, Drawer, Tool, Project, etc.) and 'unsnapshot' it. Workbench will again position the icon in the place it decides is best. For example, a disk icon will be positioned in the top right-most vacant space that Workbench can find.

Using this program is quite simple. Click on *Float*'s icon, hold the SHIFT key and double-click the icon you want to float. You'll notice some disk activity as the new information is written. Since the positioning information for that icon is buffered, you will not see an immediate change. To do so, close the window the icon is in, then reopen it. If it is a disk icon, you'll have to reboot. These icons are now free-floating again. You may also use an 'extended select' to float several icons at once. Just click on *Float*'s icon, then single click on those icons you want to float, and double-click the last of the

group. This is strictly a Workbench program and will not run correctly if called from CLI.

LaserJet Print

Laser printers are extremely useful beasts to own. At times, though, they can be intimidating and may not easily do the things you'd like or need them to do.

LaserJet Print is a toolbox full of features for your LaserJet+, LaserJet Series II, or HP LJ compatible printer.

LaserJet Print will allow you to:

- Print in landscape or portrait modes.
- Use any built-in font, cartridge fonts, and even soft fonts.
- Automatically download soft fonts into the printer.
- Do *extremely* fast printing. (This is done by totally bypassing the Preferences *printer.device*).
- Print one or *two* logical pages on a single sheet.
- Use multiple styles of page headers with date/time, file name, and page numbers.
- Adjust page length and width in inches.
- Use preset lengths/widths for paper sizes of A4, Letter, Legal, and Executive.
- Specify and select the pitch and point of a font.
- Select bold and italic fonts, plus limited support for proportional fonts (to be fixed in a later release).
- Exercise fine control over the LPI (lines per inch) setting.
- Specify the number of characters per line needed and the correct font size will be selected, if such a font exists.
- Select margins for top, bottom, left and right.
- Select tab spacing.
- Print multiple copies. (Uses printer's number of pages facility.)
- Print line numbers, either for the whole file or per page.
- Use file requesters for file selection and soft font selection (completely intuitionized).
- Call up your favourite editor directly from the program with the selected file loaded automatically.
- Use serial or parallel port.

The program has *seven* different sets of settings that can be used for different jobs as required. These can be saved and will be automatically loaded every time you run the program. A shrink window facility is provided to make the program disappear and just leave a tiny window that can be selected to have the program come up again. There is also a CLI option to have the program start up as a tiny window for all you Startup-Sequence fans.

This program gives your laser printer a lot more power.

Unfortunately, I'm not one of the lucky people who own a laser printer. I did play with a LaserJet Printer for quite a while at my dealers'. The program works as advertised and is a must have for those of you with HP LJ compatible printers.

LJ2

This is another needed utility for HP LJ compatible printers.

There are times when you want to do a draft copy of a document to check its compositional layout. Normally you would just print out the entire document, in draft mode if using a dot matrix printer.

This can be wasteful of paper and, in the case of laser printers, also wasteful of toner, which (depending on brand and model of printer) can be anywhere from $30 to $120 for a new toner cartridge.

LJ2 allows you to print two logical pages of text on the same page using a small landscape font. Two pages will print side by side on a single sheet of paper.

Each 'page' will have a title which includes the name of the file being printed, the current date and time, and the number of the page that is being printed.

The command line format is:

```
lj2  [<option>] <filename> [<destination>]
```

where **<filename>** is the name of the text file to be printed. **<Destination>** is where the output will go. The default destination is the printer selected from Preferences.

There are various options. You can disable headers, disable time and date, enable line numbering, and others.

Printer

This one I use all the time.

Printer will change your Preferences printer settings from the CLI. It allows you to load in previously configured set-ups. This is very useful for differing types of print jobs or the frequent use of different printers.

As an example, if your command line is:

```
PRINTER driver Epson port parallel pitch 10 spacing 6
```

this tells the system to use a driver called Epson, use the parallel port, set pitch to 10 and line spacing to 6 LPI.

Another example is ...

```
PRINTER load LQ_Setup
```

This will read in and set up a pre-written configuration named LQ_Setup. The 'load' keyword will search the current directory then DEVS: to find the file. If the file is not found, a file requester will be brought up.

The following is a list of valid keywords and values supported by *Printer*:

Driver	: Printer Driver Name
Port	: Serial/Parallel
Paper	: FanFold/Single
Quality	: Draft/Letter
Pitch	: 10/12/15 OR Pica/Elite/Fine
Spacing	: 6/8
Length	: 1-999
Left	: 1-999
Right	: 1-999
Size	: Letter/US-Legal/Narrow/Wide/Custom
Aspect	: Horizontal/Vertical
Shade	: B&W/GreyScale/Color
Image	: Positive/Negative
Threshold	: 1-15
Smooth	: ON/OFF
Correct	: One or more of R/G/B (e.g. RB) or None
Dither	: Ordered/Halftone/F-S
Scale	: Fraction/Integer
Density	: 1-7
Offset	: Tenths of Inches (0-255)
Center	: ON/OFF
Limits	: Ignore/Bounded/Absolute/Pixel/Multiply
Width	: Tenths of Inches OR Pixels OR Times. (0-65535)
Height	: Tenths of Inches OR Pixels OR Times. (0-65535)
Baud	: 110/300/1200/2400/4800/9600/19200/31250
Load	: Any valid Preferences file.
Save	: Any file name.
Help	: Display this message.
Show	: Shows the current printer settings.

As I said earlier, I use this one all the time. I find it much more convenient to use the load feature to change my current printer setup than to go into the printer Preferences program itself. I am not, for various reasons, a fan of ARP, but this program got me to at least put the ARP.library on my disk. *Printer* requires the ARP library (version 1.1 or higher) and Workbench version 1.3 or higher.

AutoMount

AutoMount will automatically mount all the entries in your *DEVS:mountlist* file. This will allow you to get rid of all those multiple "mount xx:" commands in your startup-sequence. The primary purpose of this program is to shorten your startup-sequence, thus shortening your boot-up time.

All you need to do is place *AutoMount* in your *c:* directory and make sure that the AmigaDOS **mount** command is there also.

If your *mountlist* is like mine, you may have several devices defined that you do not want mounted everytime you boot up. *AutoMount* takes this into account. Just add the line "/* DO NOT AUTOMOUNT*/" between the number (#) of the previous device and the name of the device you wish to ignore.

For example:

```
SPEAK:
   Handler = L:Speak-Handler
   Stacksize = 6000
   Priority = 5
   GlobVec = -1
#

/* DO NOT AUTOMOUNT */

AUX:
   Handler = L:Aux-Handler
   Stacksize = 1000
   Priority = 5
#
```

If the device to ignore is the first device in the list, add the comment on the line immediately above the device specification. *AutoMount* will not complain (and thus abort the procedure) if a device has already been mounted.

Another way to automount devices is to pass as parameters the names of the devices you wish to automount; e.g. *AutoMount* AUX: SPEAK: NEWCON: ... This method is faster than the normal approach. You just specify what you want to mount after the name of the program from the CLI or from your startup-sequence.

AutoMount has a feature specifically for floppy users. By using the keyword "FLOPPY", the program will: initialize the RAMdisk device, copy DEVS:mountlist to RAM:, copy c:mount to RAM:, and mount DEVICE FROM RAM:mountlist.

This will normally increase mount speed by 50% (this feature is limited to the 1.3 version of the AmigaDOS mount command). You can utilize the FLOPPY feature from the CLI whether the program reads your *mountlist* directly or you pass parameters to it. The command lines would be:

```
AutoMount AUX: SPEAK: NEWCON:
   or...
AutoMount FLOPPY AUX: SPEAK: FAST1:
```

Well, there you have it. I don't know how you feel about having the column feature the works of just one programmer, but I felt that Khalid has written so many fine small utilities (that I discovered only after using his *Disk Storage* program) that I thought I should pass along his work to you.

None of the programs described here are public domain. Mr. Aldoseri retains the copyright to them. However, they are freely distributable as long as his documentation files remain intact and unchanged.

He updates these programs constantly (as well as many others he's written that space limitations prevent me from telling you about - at least for this time) and is open to suggestions for other utilities that will enhance the use of the Amiga.

TxEd *Plus*™

The Text Editor for the Amiga™, *Plus* a whole lot more.

New ! V2.0 supports FFS.	**BlitzDisk**	Disk cache, speeds up floppy and hard disk reads up to 2000%.
	FastFonts	Speeds up text display.
	FunKeys	Hotkey window manipulator.
New ! Version 1.3	ARP	Latest versions of the AmigaDOS™ Replacement Programs.
	AREXX	Demo version of the AREXX, the macro processor used by TxEd Plus.

TxEd *Plus* is a powerful yet simple text editing environment designed to meet all of your text editing needs. It's small, fast, and fully configurable. The AREXX macro language connection makes TxEd *Plus* an ideal foundation for your complete multitasking system.

"As a programming and general-purpose text editor, TxEd Plus sets the standard."
-Warren Block, Sept '88 INFO Magazine

"(TxEd V1.3 is) ...a very good editor and an excellent value."
-Jan&Cliff Kent, Vol. 1 #9 Amazing Computing

"FastFonts / BlitzDisk provides much more than Facc II for a similar price."
-Warren Block, July '88 INFO Magazine

"I found the ARP commands to be smaller, faster, and more powerful than their forerunners."
-Jeff Blume, July '88 AmigaWorld

"Yes, get TxEd if you're editing almost anything on the Amiga."
-Bruce Webster, July '86 BYTE Magazine

"Still the one for me."
-Louis R. Wallace, Nov '88 AmigaWorld

$79.95

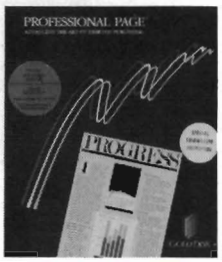
Microsmiths, Inc
PO Box 561, Cambridge MA 02140
(617) 354-1224 BIX, PLink: cheath CIS: 76004,1766

MC and Visa accepted. Mass Residents add 5% sales tax.
Amiga and AmigaDOS are trademarks of Commodore-Amiga, Inc

Creative Computers

ORDERS ONLY: 800-872-8882 OUTSIDE CA 213-370-2009 INSIDE CA

VISA, MasterCard, American Express, Discover, International Phone and Mail Orders, Educational, Corporate and Aerospace Purchase Orders Accepted.
4453 Redondo Beach Blvd., Lawndale, CA 90260 Mon.-Sat. 8 a.m.-6 p.m. PST FAX: 213-214-0932



Professional Page ($229)

SOFTWARE

3-D GRAPHICS ... 36.13
3-DEMON ... 71.95
4x4 OFF ROAD RACING ... 32.46
4TH & INCHES CONST. ... 9.95
4TH AND INCHES ... 29.24
64 EMULATOR 2, THE ... 49.95
A B ZOO ... 25.99
A-TALK III ... 64.97
A-TALK PLUS ... 49.95
AAARGH! ... 23.95
AC-BASIC COMPILER V 1.3 ... 134.06
ACCOUNTANT, THE ... 186.95
ADVENTURES OF SINBAD ... 32.46
AEGIS ANIMATOR ... 87.95
AEGIS DRAW 2000 ... 181.95
AEGIS IMAGES-PAINT ... 24.95
AESOP'S FABLES ... 19.95
ALGEBRA I ... 36.95
ALGEBRA II ... 36.95
ALIEN FIRES ... 24.95
ALL ABOUT AMERICA ... 38.95
ALTERNATE REALITY ... 27.95
AMIGA DOS EXPRESS ... 20.95
ANALYTIC ART-GRAPHICS ... 37.48
ANDROMEDA MISSION ... 25.95
ANIMAL KINGDOM ... 32.45
ANIMATE-3D ... 99.95
ANIMATION ... 47.95
ANIMATION EDITOR ... 38.95
ANIMATION MULTIPLANE ... 58.46
ANIMATION STAND ... 32.46
ANIMATOR'S APPRENTICE ... 184.95
ANNALS OF ROME ... 24.95
ARAZOK'S TOMB ... 31.25
ARCADE ACTION PACK ... 34.95
AREXX ... 32.95
ARKANOID ... 19.95
ART GALLERY FANTASY ... 23.36
ART GALLERY I ... 18.73
ART GALLERY II ... 18.73
ASHA'S FONTS ... 58.95
ASSEMPRO ... 71.95
ATREDES (BBS) ... 97.46
• AUDIOMASTER II ... 64.95
AUTODUEL ... 27.00
AWARD MAKER PLUS ... 31.15
AZTEC 68/AM-D ... 224.95
AZTEC C PROFESSIONAL ... 175.48
• AZTEC 68K CBK SPEC. ... 369.72
B.A.D. DISK OPTIMIZER ... 32.46
BAD CAT ... 30.25
BALANCE OF POWER ... 34.95
BARBARIAN ... 25.95
BARD'S TALE ... 35.95
BARD'S TALE CLUE BOOK ... 10.95
BARD'S TALE II ... 41.95
BARD'S TALE II CLUE BOOK ... 10.50
BASIC GRAMMER SERIES ... 19.48
• BATTLE CHESS ... 32.46
BATTLE SHIP ... 19.46
BBS-PC ... 62.32
BENCHMARK C LIBRARY ... 64.95
BENCHMARK IFF LIBRARY ... 64.95
BENCHMARK MODULA-2 ... 129.95
BENCHMARK SIMPLE LIB. ... 64.95
BETTER DEAD THAN ALIEN ... 21.95
BEYOND ZORK ... 33.76
BIONIC COMMANDO ... 25.97
BLACK CAULDRON ... 28.95
BLACK JACK ACADEMY ... 29.95
BLACK LAMP ... 22.00
BLACK SHADOW ... 22.95
BLITZKRIEG (1 MEG VER.) ... 38.95
BLITZKRIEG AT ARDENNES ... 34.95
BLOCKBUSTER ... 32.46
BOBO ... 23.38
BOOMERAID ... 32.46
BREACH ... 25.95
BREACH SCENARIO DISK ... 16.95
BRIDGE 5.0 ... 24.10
BUBBLE GHOST ... 22.71
BUTCHER 2.0 ... 23.13
BUTTON AND BADGE MAKER ... 38.95
C-LIGHT ... 38.95
• C-ZAR CASIO CZ ED/LIB ... 128.95
CAD PARTS FOR INTRO CAD ... 16.50
CALCULUS ... 36.95
• CALIFORNIA GAMES ... 32.46
CALLIGRAPHER ... 64.95
CAPE 68K ASSEMBLER 2.0 ... 58.47
CAPITALIZATION SERIES ... 19.46
CAPONE ... 25.95
CAPTAIN BLOOD ... 32.46
CARRIER COMMAND ... 30.95
CASINO FEVER ... 25.95
• CBM DOS 1.3 WORKBENCH ... 24.95
CELEBRITY COOKBOOK, THE ... 22.95
CENTERFOLD SQUARES ... 19.95
CHARON 5 ... 25.96
CHESSMASTER 2000 ... 32.46
CHICKEN LITTLE ... 19.95
CHRONO QUEST ... 33.95
CITY DESK ... 93.95
CITY DESK ART COMPANION ... 19.95
CITY DESK BUNDLE ... 99.95
CLIP ART #1 ... 12.95
CLIP ART #2 ... 12.95
CLIP ART #3 ... 12.95
CLIP ART #4 ... 12.95
CLIP ART #5 ... 12.95
CLIP ART #6 ... 12.95
CLIP ART #7 ... 12.95
CLIP ART #8 ... 12.95
• COMIC FONTS (NEW & NICE!) ... 59.95
COMIC SETTER ... 64.96
COMIC SETTER (FUNNY) ... 22.95
COMIC SETTER (SF DATA) ... 22.95
COMIC SETTER (SUPERHEROS) ... 22.95
COSMIC BOUNCER ... 19.95
CRAPS ACADEMY ... 25.95
CROSSWORD CREATOR ... 34.95
CRYSTAL HAMMER ... 12.96
• CURSEMASTER ... 9.95
• CYGNUS ED PROFFESIONAL ... 84.95
DARK CASTLE ... 25.95
DATA RETRIEVE ... 49.95
■ DATA RETRIEVE. PROF. ... 199.00
DEATH SWORD ... 16.95
DECIMAL DUNGEON ... 32.46
DEEP SPACE ... 19.96
DEFCON 5 ... 25.95
DEFENDER OF THE CROWN ... 34.95
DEJA VU ... 34.95
DELUXE HELP CALLIG. ... 22.71
DELUXE HELP DIGIPAINT ... 21.84
DELUXE HELP DPAINT II ... 21.84
DELUXE HELP PHOTON ... 21.95
DELUXE MAPS ... 16.95
DELUXE MUSIC 2.0 ... 69.95
DELUXE PAINT II ... 89.95
DELUXE PHOTO LAB ... 99.95
DELUXE PRODUCTIONS ... 139.95
DELUXE VIDEO 1.2 ... 89.95
DEMONSTRATOR, THE ... 21.95
DESIGN DISK:
ARCH SCULPT ... 22.74
ARCH VSCAPE ... 22.74
FUTURE SCULPT ... 22.74
FUTURE VSCAPE ... 22.74
HUMAN SCULPT ... 22.74
HUMAN VSCAPE ... 22.74
MICROB SCULPT ... 22.74
MICROB VSCAPE ... 22.74
DESTROYER ... 25.95
DEVELOPERS TOOLKIT ... 36.95
FAST ACCESS WRD. PERF. ... 12.95
FERRARI FORMULA ONE ... 33.95
FINAL ASSAULT ... 32.46
FINAL MISSION ... 19.95
FINANCIAL COOKBOOK ... 14.95
FINE PRINT ... 41.95
FIRE 'N FORGET ... 25.95
FIREPOWER ... 16.95
FIRST LETTERS & WORDS ... 33.00
FIRST SHAPES ... 33.00
FISH ... 29.95
FLEET CHECK ... 25.96
FLIGHT SIMULATOR II ... 37.46
KEY TO "C" ... 22.95
KICK-WORK 1.3 ... 22.95
KIDTALK ... 27.95
KIKUGI ... 18.95
KINDERAMA ... 32.46
KING OF CHICAGO ... 34.95
KNIGHT ORC ... 30.95
■ LATTICE "C" 5.0 COMPLETE! ... 199.00
LATTICE C 4.0 ... 162.95
LATTICE C++ ... 357.95
LAZERSCRIPT ... 28.10
LEADER BOARD DUAL PACK ... 22.72
LEADER BOARD TORNA DISK ... 13.95
MURAL PRINTER ... 29.95
MUSIC MOUSE ... 51.95
MUSIC STUDENT ... 37.95
NAG PLUS 3.0 ... 49.95
NEWSLETTER FONTS COLOR ... 29.95
OBLITERATOR ... 25.95
OFFSHORE WARRIOR ... 25.96
ONLINE 2.0 ... 43.56
OPERATION CLEAN STREETS ... 25.95
OTG DSM (DISASSEMBLER) ... 43.95
P.O.W. ... 25.95
PAGE FLIPPER PLUS F/X ... 103.95
PAGESETTER ... 93.72
PALADIN ... 25.95
PALADIN (QUEST DISK 1) ... 16.88
PAWN, THE ... 16.21
PCLO PRNT. CUR. LAYOUT ... 199.95
PCLO+ PRNT. CUR. LAYOUT ... 399.95
PHANTASIE ... 27.47
PHANTASIE 3 ... 24.95
• PHASAR-FIN'L MGMT ... 62.48
• PHOTON CELL ANIMATOR ... 89.95
PHOTON PAINT ... 64.96
PHOTON PAINT EXP. DISK ... 19.95
PHOTOSYNTHESIS ... 97.95
PINBALL I.O. ... 19.46
PINK PANTHER ... 28.95
PIONEER PLAGUE ... 25.96
PIXMATE ... 45.95
PLANET PROBE ... 19.95
PLATOON ... 29.21
PORTS OF CALL ... 29.95
POWER WINDOWS 2.5 ... 58.47
POWERSTYX ... 22.95
PRE CALCULUS ... 36.95
PRESIDENT IS MISSING ... 25.95
PRIME TIME ... 25.95
PRINTMASTER PLUS ... 31.23
PRO MIDI STUDIO ... 130.95
PRO VIDEO FONT SET #1 ... 84.95
PRO VIDEO FONT SET #2 ... 84.95
PRO VIDEO FONT SET #3 ... 84.95
• PRO VIDEO PLUS ... 184.95
PROBABILITY THEORY ... 36.95
• PROFESSIONAL PAGE ... 229.00
PROJECT D ... 31.95

AMIGA MUSIC PACK

- SHS-10 YAMAHA KEYBOARD
- MIDI INTERFACE
- DELUXE MUSIC INSTRUCTION SET
- "IT'S ONLY ROCK & ROLL" SOFTWARE

ALL FOR A LOW PRICE OF $179! (with purchase of computer $149)

DEVPAC AMIGA ASSEM. ... 129.95
DIGI-PAINT ... 41.95
DIGI-VIEW ... 143.95
DIGI-VIEW UPGRADE ... 13.95
• DIRECTOR'S TOOLKIT ... 24.95
DIRECTOR, THE ... 45.95
DISCOVERY EXP. DISKS ... 12.97
DISCOVERY/MATH ... 25.00
DISCOVERY/SPELL ... 25.00
DISCOVERY/TRIVIA ... 25.00
DISCRETE MATHEMATICS ... 36.95
DISK MECHANIC, THE ... 58.50
DISK TO DISK ... 34.95
FLOW ... 62.32
FONTS AND BORDERS ... 22.95
• FORMS IN FLIGHT II ... 79.95
FORMULA ONE AMIGA ... 19.46
FOUNDATIONS WASTE ... 23.62
FRACTION ACTION ... 32.46
FROST BYTE ... 19.46
GALACTIC INVASION ... 16.95
GALILEO 2.0 ... 45.46
GALLERY 3-D (SCULPT) ... 51.95
GEOMETRIC LIBRARY ... 11.95
GETTYSBURG ... 38.95
GIZMOZ 2.0 ... 39.95
LEADER BOARD-GOLF GAME ... 27.00
LEARNING CURVE ... 51.95
LEARNING THE ALPHABET ... 19.46
LEATHERNECK ... 25.95
LEISURE SUIT LARRY ... 26.40
LIFE CYCLES AMIGA ... 19.46
LIGHTS! CAMERA! ACTION! ... 59.95
LINKWORD DUTCH ... 20.55
LINKWORD FRENCH ... 20.55
LINXWORD GERMAN ... 20.55
LINKWORD GREEK ... 20.55
LINKWORD ITALIAN ... 20.55
LINKWORD PORTUGUESE ... 20.55
LINKWORD RUSSIAN ... 20.55
LINKWORD SPANISH ... 20.55
LINT (GIMPLE) ... 63.70
LITTLE RED HEN ... 19.95
LOGIG WORKS ADVANCED ... 189.95
LOTTERY MAGIC ... 19.95
• M INTELLIGENT MUSIC ... 143.46
• MAGELLAN ... 119.00
MAGICAL MYTHS ... 32.46
MAGICIAN'S DUNGEON ... 21.95
MAJOR MOTION ... 25.96
MANIAX ... 16.95
MARAUDER II-DUPLICATION ... 27.47
MARBLE MADNESS ... 33.00
MASTERING CLI ... 25.95
MASTERPIECE FONT ... 167.95
MATCH IT ... 25.95
MATH MAGICIAN ... 27.49
MATH TALK ... 31.95
MATH TALK FRACTIONS ... 24.95
MATH WIZARD ... 32.46
MATH-AMATION ... 64.95
MAVIS BEACON TYPING ... 30.89
MAXIPLAN 500 ... 93.95
MAXIPLAN PLUS ... 124.95
MEAN 18 COURSE DISK ... 14.96

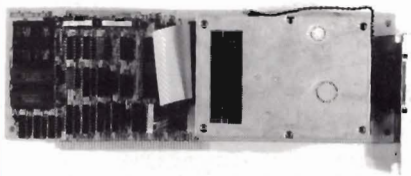
Excellence! ($159)

PROSCRIPT ... 32.95
PROWRITE 2.0 ... 78.10
PUBLISHER PLUS ... 99.95
PUNCTUATION SERIES ... 19.46
QRADRALIEN ... 23.61
QUARTERBACK ... 45.95
QUESTRON II ... 32.46
QUIZ MASTER ... 49.95
R.R. AESOP'S FABLES ... 19.95
RAW COPY 1.3 ... 38.95

We carry OVER 1000 items
The best NEW products
The LOWEST PRICES

NEW PRODUCTS ARRIVE DAILY. CALL FOR UNLISTED ITEMS.

DISK WICK ... 32.46
DISKMASTER ... 37.95
DOMINOES ... 16.95
DOS TO DOS ... 37.95
DOUG'S MATH AQUARIUM ... 51.95
DOUGS COLOR COMMANDER ... 19.46
DR. T'S 4-OP ED/LIB ... 92.95
DR. T'S BACH SONGBOOK ... 17.95
DR. T'S CASIO V21 ED/LIB ... 92.95
DR. T'S COPYIST ... 155.36
DR. T'S CZ RIDER ... 92.95
DR. T'S D-100 ED/LIB ... 92.95
DR. T'S D-110 ED/LIB ... 92.95
DR. T'S D-50 ED/LIB ... 92.95
DR. T'S DR. DRUMS ... 17.95
DR. T'S DR. KEYS ... 17.95
DR. T'S DX7 ED/LIB ... 92.95
DR. T'S ESO-APADE ESO-1 ... 92.95
DR. T'S KAWAI K-1 ED/LIB ... 92.95
DR. T'S KAWAI K-5 ED/LIB ... 92.95
DR. T'S KCS ... 154.95
DR. T'S LEXICON ED/LIB ... 92.95
DR. T'S MIDI REC. STUD ... 43.50
DR. T'S MT-32 ED/LIB ... 92.95
DR. T.S MATRIX-6 ED/LIB ... 92.95
• DRAGON'S LAIR ... 46.50
DREAM ZONE ... 32.46
DRUM STUDIO ... 32.47
• DUNGEON MASTER ... 32.46
DUNGEON MASTER HINTS ... 12.95
DYNAMIC DRUMS ... 49.95
DYNAMIC STUDIO ... 139.00
DYNAMIC-CAD ... 299.00
EARL WEAVER BASEBALL ... 34.95
EASY LOANS ... 24.95
EBON STAR ... 25.96
EMETIC SKIMMER ... 25.95
EMPIRE ... 34.32
ENCORE ... 45.95
EUROPEAN SCENERY DISK ... 17.95
• EXCELLENCE ... 159.00
EXPRESS PAINT ... 90.95
EXTEND ... 25.95
FACC II ... 21.95
FAERY TALE ADVENTURE ... 31.95
FAERY TALE GUIDEBOOK ... 7.95
FALCON ... 32.46
• FAMILY TREE ... 32.46
FANCY 3D FONTS ... 46.95
FANTAVISION ... 42.95
GLOBAL COMMANDER ... 27.48
GOLD DISK FONT SET #1 ... 21.85
GOLD SPELL II ... 29.95
GOLDEN PATH ... 29.95
GOMFI 3.0 (SOFTWARE) ... 27.95
GRABBIT ... 20.95
GRADE MANAGER ... 58.95
GRAND SLAM TENNIS ... 31.25
GRAPHICS STUDIO, THE ... 38.95
GREAT STATES II ... 25.95
• GRETZKY HOCKEY ... 32.46
GUILD OF THIEVES ... 30.95
HAROBALL ... 28.95
HARRIER COMBAT SIM ... 32.95
HARRIER MISSION ... 16.95
HEAD COACH ... 32.46
HEROES OF THE LANCE ... 27.95
HIT DISK VOL. I ... 32.46
HOLE IN ONE ... 25.99
HOLLYWOOD POKER ... 25.95
HOME BUILDERS CAD ... 129.95
HUNT FOR RED OCTOBER ... 27.46
HYBRIS ... 25.99
ICON PAINT ... 58.95
IMPACT-BUS. GRAPHICS ... 62.46
IMPOSSIBLE MISSION II ... 32.46
INDOOR SPORTS ... 31.22
INOVA TOOLS ... 52.03
INSANITY FIGHT ... 25.95
INSTANT MUSIC 1.2 ... 33.00
INTELLITYPE ... 35.95
INTERCEPTOR ... 37.95
INTERCHANGE ... 29.95
INTERCHANGE CONVERSION ... 16.95
INTERCHANGE OBJECTS #1 ... 16.95
INTERCHANGE TSLV MOD. ... 12.96
INTERFONT ... 75.38
INTERNATIONAL SOCCER ... 25.96
INTO THE EAGLE'S NEST ... 28.95
INTROCAD ... 49.95
• INVISION ... 99.95
IT'S ONLY ROCK & ROLL ... 21.95
J FORTH ... 77.96
JET ... 37.46
JET SET FONT SET ... 32.50
JINXTER ... 25.95
JOKER POKER ... 32.46
KAMPFGRUPPE ... 41.95
KARA FONTS ... 54.95
KARATE KID II ... 25.95
MEAN 18 GOLF ... 28.95
MEDIALINE FONTS ... 22.95
MENACE ... 19.98
METACOMCO ASSEMBLER ... 68.95
METACOMCO PASCAL 2.0 ... 92.95
METACOMCO SHELL ... 39.95
METACOMCO TOOLKIT ... 34.95
METASCOPE DEBUGGER ... 59.95
MICROFICHE FILER ... 69.95
MICROFICHE FILER PLUS ... 116.95
MICROLAWYER ... 37.95
MISSION CON-BAT ... 31.18
MODELER 3-D ... 85.00
MOEBIUS ... 39.95
MONEY MENTOR C VERSION ... 59.98
MOVIE SETTER ... 64.95
MULTI-FORTH ... 59.95
MULTI-PREFS ... 19.45
REACH FOR THE STARS ... 27.48
READ & RHYME ... 31.95
READ-A-RAMA ... 32.46
REALM OF THE WARLOCK ... 22.74
REBEL CHRG. CHICKAMANGA ... 39.95
• REEL FISHIN'—GONE FISHIN' ... 30.88
RETURN TO ATLANTIS ... 34.95
ROAD RAIDERS ... 25.96
ROADWAR 2000 ... 27.47
ROADWAR EUROPA ... 29.95
ROADWARS ... 23.95
ROB. READER UGLY DUCK ... 19.45
ROBBERY ... 16.21
ROCK CHALLENGE ... 25.95
ROCKET RANGER ... 32.46
ROCKFORD ... 23.95
ROMANTIC ENCOUNTER ... 25.95
SCENERY DSK #7 E. COAST ... 18.71



GVP GREAT VALLEY PRODUCTS

For Amiga 2000:
IMPACT HC/45—48 MB 28ms Hard Card ... 799.00
IMPACT HC/40Q—42 MB 11ms Hard Card ... 799.00
IMPACT HC/80Q—84 MB 11ms Hard Card ... 1199.00
For Amiga 500:
IMPACT A500-SCSI/HD20—20 MB HD ... 749.00
IMPACT A500-SCSI/HD45—45 MB HD ... 995.00
IMPACT A500-SCSI/HD40Q—42 MB 11ms High-Speed Hard Drive ... 1049.00
IMPACT A500-SCSI/HD80Q—84 MB 11ms High-Speed Hard Drive ... 1449.00
Quantum 80S, 84 MB 11ms 3.5" HD ... 995.00
Shock mounted, 64KB cache (for 11ms speed), SCSI interface. Compatible with IMPACT or A2090 boards.
Quantum 40S—42 MB version of above drive ... 599.00

OTHER CREATIVE PROMOTIONS

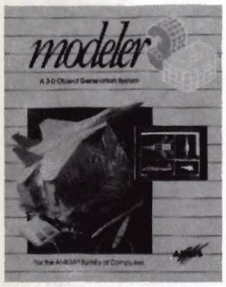
NEW AMIGA 2500 COMPUTER AND 2620 TURBO CARD! CALL FOR OUR LOWEST PRICE.

AMIGA 500 and 2000 CPU's and package deals available ... CALL FOR DETAILS!

- WordPerfect: The best non-graphic wordprocessor on the Amiga. $399 retail. Our price? ONLY $179!!!
- FREE VCR with purchase of computer—call for details.
- Package pricing on computers available—call for details.
- Amiga Live! 2000: $349
- Flicker Fixer! (A2000 only) — *Eliminate all flicker and scanline gaps 100% — optional multisync monitor: $479*

WE CAN BEAT ANY ADVERTISED PRICE! BUT WE SELDOM HAVE TO — OUR PRICES ARE THE LOWEST! (IF YOU SEE A LOWER PRICE ON AN ITEM, GIVE US A CALL.)

SCENERY DSK #11 E. COAST 18.72
SCHEMATIC CAPTURE 199.95
SCRABBLE 27.46
SCRIBBLE 62.32
SCULPT 3-D 69.95
SCULPT 4-D 399.95
■ SDI: CINEMAWARE SERIES 34.95
SECONDS OUT 19.46
SECURITIES ANALYST 49.95
SERIAL EXPANDER 15.95
SEX VIXENS 25.95
SHADOW GATE 31.95
SHAKESPEARE 146.95
SHANGHAI-STRATEGY GAME 27.46
SILENT SERVICE 25.95
SINBAD & FALCON 34.95
SKY BLASTER 19.95



AEGIS Modeler 3D ($65)

SKY CHASE 26.95
SKYFOX II 27.45
SLAYGON 19.95
SMOOTH TALKER 33.95
SNAKE PIT 24.95
SOCIAL STUDIES 19.46
SOFTWOOD FILE SG 78.21
SONIX (AEGIS) 49.95
SOUND DASIS 64.95
SOUNDLAB 209.95
SOUNDQUEST:
CZ MASTER 92.95
D-10 MASTER 92.95
D-50 MASTER 92.95
D-110 MAST. 92.95
DX7 MASTER 109.95
DX7-II MASTER 109.95
ESQ/ESQ-80 MST 92.95
GENERIC MASTER 77.95
K-1 MASTER 92.95
M-1 MASTER 92.95
MT-32 MASTER 92.95
TEXTURE 92.95
TX-802 MASTER 92.95
TX-81Z MASTER 92.95
SOUNDSCAPE PATTERN SPLAT. 35.95
SOUNDSCAPE UTILITIES 1 ..... 35.95
SOUNDSCAPE UTILITIES II 42.95

TALES ARABIAN NIGHTS 32.46
TALKING ANIMATOR, THE 39.00
TALKING COLORING BOOK 18.95
TANGLE WOOD 25.95
TARGIS 28.95
TECHNOCOP 32.46
TELE EPIC 25.96
TELE WARS II 25.96
TELEGAMES 23.95
TELEWARS 24.95
TERA QUESTRON 25.95
TERRAIN 19.95
TERRORPODS 25.95
TEST DRIVE 34.95
TETRA QUEST 25.95
TETRIS 21.80
TEXTCRAFT PLUS 64.93
THEXDER 23.95
THREE LITTLE PIGS 19.95
THREE STOOGES 35.95
THUNDERBOY 22.95
TIME & MAGIK 26.95
TIME BANDITS 21.95
TOOL CADDY 32.46
TOWER TOPPLER 32.46
TRANSCONT. RAILROAD 25.95
TRIGONOMETRY 36.95
TRUE BASIC 68.95
TURBO PRINT 32.95
• TURBO SILVER 129.95
TV SHOW 64.95
• TV SPORTS FOOTBALL 34.95
TV TEXT 62.32
TXED PLUS 51.95
ULTIMA III 28.95
ULTIMA IV 38.96
ULTRA DOS 37.95
UMS CIVIL WAR DATA DISK 12.95
UMS VIETNAM DATA DISK 12.95
UNINVITED 34.95
UNIVERSAL MILITARY SIM. 32.46
VAMPIRE'S EMPIRE 29.95
• VIDEO EFFECTS 3D 129.95
VIDEO VEGAS 24.10
VIDEO VISIONS 1 ..... 12.95
VIDEO VISIONS 2 ..... 12.95
• VIDEOSCAPE 3D 124.95
VIDEOTITLER 99.95
VIP VIRUS 32.46
VIRUS (GAME) 19.46
VOCABULARY IMPROVE 25.95
W SHELL 31.95
WARLOCK 22.71
WBEXTRAS 24.95
WHO FRAMED ROGER RAB. 29.95
WINDOW PRINT II 22.95
WINTER CHALLENGE 9.95
WINTER GAMES 27.46
WORD MASTER 32.46
WORD PERFECT 179.00
WORD PERFECT LIBRARY 84.95
WORKS, THE 124.95
WORLD CLASS LEADER BRD 25.97
WORLD GAMES 27.46
WORLD TOUR GOLF 27.45
WRITE 'N FILE 59.95
WSHELL 32.46
X-CAD 389.00
ZAK MCKRACKEN 29.95
ZERO GRAVITY 19.46

HOT NEW PRODUCTS

(As featured in this magazine)

VIDEO & AUDIO:
- MidiVU
- C-Zar
- Comic Fonts
- Pro-Sound Designer
- MIDI Magic
- Digi View Gold
- Zoetrope
- Photon Cell Animator
- Sculpt-Animate 4D

GAMES:
- Operation Wolf
- Bubble Bobble
- Rastan
- Dragon's Lair
- TV Sports Football
- Lords of the Rising Sun
- Cosmic Bouncer
- Rock Challenge
- Univeral Military Simulator
- Charon 5
- Joker Poker
- Advanced Dungeons & Dragons
- Final Assault
- Evil Garden
- Mission: Con-Bat
- Autodual
- Tower Toppler
- TeleEpic & TeleWar II

PRODUCTIVITY:
- Lattice C++
- Publishing Partner Professional
- Professional DataRetrieve
- ComicSetter
- Professional Draw
- Draw 2000
- Superbase Personal 2
- Superplan
- Deluxe Print II

UTILITIES:
- B.A.D. (hat floppy and hard disk acceleratar)

ACCESSORIES:
- CMI MIDI 1
- INTRUDER ALERT

PLEASE CALL FOR PRICES

SOURCE LEVEL DEBUGGER 49.96
SPACE CUTTER 19.95
SPACE MATH 29.95
SPACE RACER 19.46
SPACE STATION OBLIVION 32.46
SPACEQUEST 33.00
SPELLBOUND 25.95
SPELLER BEE 31.25
SPELLING IMPROVEMENT 25.95
STAR WARS (EMPIRE) 30.25
STARGLIDER 30.95
• STARGLIDER II 29.95
STARRAY 23.61
STELLAR CONFLICT 28.95
STELLARYX 25.95
STOCK MARKET-THE GAME 16.95
STREET CAT 16.95
STREET SPORTS BSKBALL 13.95
STRIP POKER 27.46
STRIP POKER DATA #4 ..... 12.95
STRIP POKER DATA DISK #5 ..... 12.95
STRIP POKER II 25.95
STUDIO FONTS V1 (COLOR) 29.95
STUDIO MAGIC 64.95
SUB BATTLE 24.95
SUMMER CHALLENGE 9.95
SUPERBACK 51.95
SUPERBASE 49.95
SUPERBASE PERS. II 94.48
• SUPERBASE PRO. 3.0 ..... 229.95
SUPERSTAR ICE HOCKEY 29.86
SUPERSTAR ICE HOCKEY 32.46
SURGEON, THE 31.95
SWORDS OF SODAN 32.46
SYNTHIA 59.95
T SHELL 32.46

ZOETROPE 89.95
ZOOM 21.95
ZORK TRILOGY 47.95
ZUMA FONTS VOL 1-4 ..... 68.95
ZYNAPS 23.95

HARDWARE

68020-12 CPU/PROCESSOR 170.00
68030-16/20 CPU/PROCESSOR 570.00
68881 CRYSTAL OSSC. 16MHZ 5.00
68881-16 MATH COPROCESSOR 200.00
68881-20 MATH COPROCESSOR 300.00
68882-20 MATH COPROCESSOR 500.00
68882-25 MATH COPROCESSOR 620.00
AIR DRIVE 3.5" EXT DRIVE 169.00
AIR DRIVE 3.5" INT A2000 ..... 149.95
AMAS MIDI SAMPLER 109.95
AMIGEN GENLOCK 149.00
AMPLIFIED SPEAKERS 2x50 W 84.95
APRO DRAW 12x12 NO CURS. 464.95
APRO DRAW 12x12 W/CURSOR 499.95
BOING OPTICAL MOUSE 114.95
BYTE BOX 0K-RAM OPTIONAL 249.00
CA-880 FLOPPY DRIVE 159.00
CBM A2300 INT GENLCK 2000 ..... CALL
CBM A520 COMPOS. ADAP 500 ..... 49.95
CBM AMIGA MOUSE 65.00
• CBM DOS 1.3 ROM 39.95
CHEETAH MK5 MIDI KEYBRD 235.00
CLTO TIMESAVER ..... 57.
CMI MIDI I (EXT) 52.6
COLOR SPLITTER 79.95
COMMODORE COLT PC 640K 599.00
CPS 500-POWER SUPPLY A500 ..... 74.95
CSA DRAGSTRIP RAM ACCEL. 723.45

CSA FASTRACK 68020 ACCEL. 905.45
DELUXE MIDI INTERFACE 59.95
DESKJET 128K RAM CART 117.95
• DIGI VIEW GOLD A500/A2000 ..... 149.00
EASYL 1000 TABLET 369.00
EASYL 2000 TABLET 369.00
EASYL 500 TABLET 369.00
ECE MIDI 1000 ..... 48.95
ECE MIDI 500/2000 ..... 48.95

Boing! Mouse

Professional Optical 3-button mouse for your Amiga. Available EXCLUSIVELY at Creative Computers!

The first and only Optical Mouse for the Amiga — $114
This mouse is definitely the best mouse you can buy for your Amiga.

- Optical technology eliminates friction and momentum coefficients for super-precise handling.
- No moving parts for increased reliability.
- Middle button makes it the only mouse compatible with A2024 software and X Windows.
- Compatible with ALL Amiga computers.

ONLY at CREATIVE COMPUTERS

EXP-1000 1M UNPOPULATED 220.00
• FLICKER FIXER (HARDWARE) 479.00
FRAMEGRABBER 519.00
GEN ONE GENLOCK 739.00
GOMFI 3.0 W/BUTTON 46.95
GVP A2000 AUTOBDDT EPROM 29.95
GVP A500 RAM/ROM MODULE 69.95
HURRICANE 68030 ADAP. BRD 300.00
HURRICANE A1000 ACCEL BRD 350.00
HURRICANE A1000 MEM. BRD 540.00
HURRICANE A2000 ACCEL BRD 500.00
HURRICANE A2000 MEM. BRD 540.00
HURRICANE ACCEL/MEM. BRD. 1060.00
HURRICANE DRAM, 1 MEG 525.00
IMPRINT FOR POLAROID PAL. 399.00
INTRUDER ALERT 124.02
LASERJET-2 RAM, 1 MEGBYTE 419.95
LENS, 16 MM, NO IRIS F1.6 ..... 21.85
LENS, 16 MM, W/IRIS F1.6 ..... 66.25
LIGHT PEN 97.95
LIVE! 500 ..... 289.00
LIVE! 1000 ..... 370.00
• LIVE! 2000 ..... 349.00
MAGNI 4084 GENLOCK 1595.00
MICROBT. 4 x POP-SIMM BRD 74.50
MICROBT. 8-UP RAM BRD, 0K 145.00
MICROBT HARDFRAME SCSI 239.00
MICRON CAGE FOR A500 ..... 199.95
MIDI GOLD 2000 (INT) 68.47
MINISCRIBE 20 MB, 40 MS 295.00
MINISCRIBE 40 MB, 28 MS 615.00
NEC P2200 PRINTER 400.00
NERIKI GENLOCK 1975.00
OKIMATE 20 W/PLUG N PRNT 249.00
OKIMATE PLUG N PRINT 79.95
OVERDRIVE HD CONTROLLER 199.95
PANASONIC WV1410 CAMERA 225.00
PANASONIC WV1500 CAMERA 320.00
PEOPLE METER BID FEEDBACK 39.95
PERFECT SOUND DIGITIZER 67.95
• PERFECT VISION 169.95
PRO SOUND DESIGNER 119.95
PROCESSOR ACCEL. CMI 159.00
PROGEN 379.00
QUANTUM 80 MB DRIVE, 11MS 595.00
• ROLAND MT-32 TONE MODULE 499.00
SPIRIT 0 MB FOR A1000 ..... 249.00
SPIRIT 0 MB FOR A500 ..... 199.00
STAR NX1000 PRINTER 199.00
STAR NX1000 RAINBOW 279.95
STARBOARD 2 SCSI MODULE 100.00
STARBOARD 2 UPPER DECK 80.00
STARBOARD 2/A1000 0K 250.00
STARBOARD 2/A500 0K 300.00
STARBOARD SB2000 ADAPTER 42.00
STEREO SOUND SAMPLER STRV 87.95
SUPERSYSTEM 500 ..... 200.00
■ SUPERGEN 649.00
• SUPRA 2400 MODEM 149.00
SUPRA DRIVE 20 MEG A1000 ..... 699.00
SUPRA DRIVE 20 MEG A500 ..... 699.00
SUPRA DRIVE 30 MEG A1000 ..... 799.00
SUPRA DRIVE 30 MEG A500 ..... 799.00
TRUMP CARD SCSI CNTR 2000 ..... 168.95
TWIN-DRIVES EXT 3.5" FDD 279.00
UNIDRIVE 3.5" EXT. DRIVE 159.00
VI 2000 VIDEO INTERFACE 69.95
VI 2000 VIDEO/RF MODULAT. 79.95
VI 500 VIDEO INTERFACE 59.95
VI 500 VIDEO/RF MODULATOR 69.95
X-SPECS 3D 99.95
XEROX 4020 INK JET CDLOR 1140.00
XEROX 4020 STARTER KIT 156.00

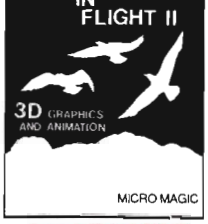
ACCESSORIES

A1084 CARRING CASE 51.95
A500 & MONITOR CARRY BAGS 69.95
AMIGA BAG 29.95
AMIGA BRASS KEY RING 3.95
AMIGA FASHION GOLF SHIRT 19.95
AMIGA FASHION JACKET 39.50
AMIGA MUG 5.78
AMIGA NOTE PADII 4.99
AMIGA PEN & PENCIL SET 19.95
AMIGA PLASTIC VISOR 1.95
AMIGA SLEEVE TSHIRT 7.95

AMIGA T-SHIRT 7.95
AMIGA TERRY CLOTH VISOR 5.50
AMIGA TOWEL 15.95
AMIGA TWILL CAP W/ CORD 7.95
BONUS DISKETTES, VERBATIM 13.95
CABLE, 10' CAMERA 12.95
CABLE, 6' MIDI LINK 7.95
CABLE, AMIGA TO NEC II 24.95
CABLE, AMIGA TO SONY 1302 ..... 19.95
CABLE, AMIGEN TO 1084 ..... 19.95
CABLE, CENTR. TO CENTR. 19.95
CABLE, DB25M-DB25F 8 FT. 16.95
CABLE, DRIVE EXT. 2 FT. 19.95
CABLE, DRIVE EXT. 3 FT. 29.95
CABLE, FLCK FIX TO NEC II 19.95
CABLE, FLCK FIX-SONY 1302 ..... 19.95
CABLE, IMAGEWRITER A1000 ..... 22.95
CABLE, IMAGEWRITER A2000 ..... 22.95
CABLE, MONITOR EXT 4FT 21.95
CABLE, MOUSE EXTENSION 9.95
CABLE, PARALLEL A1000 ..... 19.00
CABLE, PARALLEL A500/2000 ..... 19.00
CABLE, SERIAL A1000 ..... 19.00
CABLE, SERIAL A500/A2000 ..... 19.00
CLEANING KIT (ALLSDP) 14.95
CLEANING KIT SM. ALLSDP 6.95
CONTROL CENTRE FOR A500 ..... 71.95
COPY/CAMERA STAND,LIGHTS 64.95
CURTIS 3.5" DISK FILE 30 ..... 7.95
CURTIS DIAMOND 33.95
CURTIS DIAMOND PLUS 46.95
CURTIS EMERALD 40.95
CURTIS PHONE LINE SURGE 12.36
CURTIS RUBY 58.95
CURTIS SAPPHIRE 56.95
DESKJET BLACK CARTRIDGE 15.95
DESKJET COURIER FONT CRTG 62.95
DESKJET DUST COVER 29.95
DESKJET EPSDN FX-80 EMUL. 62.95
DESKJET HELV 12 POINT 62.95
DESKJET HELV 14 POINT 62.95
DESKJET HELV 8 & 10 PT. 79.95
DESKJET HELV ASCII FONTS 104.95
DESKJET LANDSCAPE FONT 79.95
DESKJET LANDSCAPE UPGRADE 79.95
DESKJET LETTER GOTHIC 79.95
DESKJET PRESENTATION FONT 79.95
DESKJET PRESTIGE ELITE 79.95

AmiGen ($149)

DESKJET RMN/HELV SOFT FNT 79.95
DESKJET STAND/ORGANIZER 84.95
DESKJET TMS RMN 12 POINT 62.95
DESKJET TMS RMN 14 POINT 62.95
DESKJET TMS RMN 8 & 10 PT 79.95
DESKJET TMS RMN ASCII 104.95
DIGI-DROID 69.95
DISK FILE 80 ..... 19.95
DUST COVER FOR A2000 ..... 11.95
DUST COVER FOR A500 ..... 8.95
EPYX 500XJ JDYSTICK 14.95
ERGO STICK 17.95
FLICKER MASTER 13.95
FOUR PLAYER ADAPTER 11.95
FUJI DISKETTES 19.95
G.E. SURGE PRD EPC-6NS 34.95
GAME PLAY COMPACT DISK 12.95
GENDER BENDER (DIGI-VIEW) 20.95
JOYSTICK Y ADAPTER 10.95
KEYBOARD SKIN A2000 ..... 19.46
KEYBOARD SKIN A500 ..... 19.46
KEYBOARK SKIN A1000 ..... 19.46
LASERJET-2 TONER CARTRIDGE 104.00
LASERJET-2 8.5 x 14 TRAY 104.95
LASERJET-2 COURIER FONTS 144.50
LASERJET-2 ENVELOPE TRAY 79.95
LASERJET-2D ENVELOPE TRAY 279.95
LIGHT GUN 48.95
MICHTRDN DISK FOLDER 10.95
MIDI MAGIC 97.95
MINO LIGHT 7 ..... 164.95



Forms In Flight II ($79.50)

MOUSE MATS 6.95
MOUSE TOP 9.95
PAINTJET BLACK CARTRIDGE 23.60
PAINTJET COLOR CARTRIDGE 29.10
PAINTJET CUT-SHEET PAPER 19.10
PAINTJET DUST COVER 19.95
PAINTJET PRINTER STAND 66.95
PAINTJET TRANSPARENCY FILM 54.05
PAINTJET Z-FOLD PAPER 14.95
POWERPLAYER JOYSTICK 17.95
• RAM, 1 MB (256K-15 CHIPS) 352.00
RAM, 1 MEG x 1 -10 CHIPS 37.50
RAM, 1.5MB (256-15 CHIPS) 528.00
RAM, 1MB (1 MEG-10 CHIPS) 300.00
RAM, 256K x 1 -12 CHIPS 12.00
RAM, 256K x 1 -15 CHIPS 11.00
RAM, 512KB (256-12 CHIPS) 192.00
RAM, 512KB (256-15 CHIPS) 176.00
SONY DISKS 10 PACK 20.95

SPEED MOUSE PAD 9.71
TEAK-TECH 120 DISKHOLDER 32.95
TEAK-TECH 60 DISKHOLDER 27.95
TEAK-TECH 84 DISKHOLDER 24.95

TERMINATOR (JOYSTICK) 15.95
TRACKBALL EMULATES MOUSE 39.95
TRANSPORT CONTROLLER PHO 194.95
XEROX 4020 4-PACK INK 19.95

FREE SHIPPING!*

CREATIVE COMPUTERS is both a mail order company with a store's support and three store showrooms with mail order prices. Visit one of our showrooms today. You'll be amazed!

3 RETAIL SHOWROOM LOCATIONS TO SERVE YOU

SANTA MONICA & WESTSIDE: 318 Wilshire Blvd., Santa Monica, CA 90401. Tues.-Sat. 11-7 p.m./Sun. 11-5 p.m./Phone: 213-394-7779

LAWNDALE & SOUTH BAY: 4453 Redondo Beach Blvd., Lawndale, CA 90260. Mon.-Sat. 11-7 p.m./Phone: 213-542-2292

VENTURA, OXNARD & SANTA BARBARA: 2112 E. Thompson Dr., Ventura, CA 93001. Tues.-Sat. 11-7 p.m./Sun. 12-5 p.m./Phone: 805-652-0325

COMMODORE AMIGA™

THE CREATIVE COMPUTERS ADVANTAGE:
- Authorized Amiga Dealer
- The largest dealer of Amiga products in the World
- Three store locations means excellent support
- Amiga-specific—unlike the competition, we don't claim being Amiga specific while selling other brands under another business name
- Authorized service center
- Uniform low pricing and largest selection, no hidden costs or catches
- We don't charge your cord until the product ships

CUSTOMER SERVICE: Call 213-542-2292 from 11 a.m. to 7 p.m. PST for technical questions.

VISA · MasterCard · AMERICAN EXPRESS Card · DISCOVER

***SHIPPING INFO:** No surcharge for Visa/MasterCard; 4% surcharge for American Express; 1% surcharge for Discover. Shipping within the continental United States for software orders over $100 (U.S.) is FREE. Please call for all other shipping information. Educational, corporate ond aerospace purchase orders accepted.

RETURN POLICY: Defective merchandise under warranty will be repaired or replaced. Returned products must be in original condition and packaging. No refund on defective products or products that do not perform satisfactorily. We make no guarantees for product performance.

CONDITIONS: Creative Computers reserves the right to limit the sales of ony items to local in-person pick-up only. Prices and availability subject to change without notice.

Creative Computers

CREATIVE COMPUTERS GOES INTERNATIONAL!

TOLL-FREE NUMBERS: Minimum orders $100 (U.S.). Orders only please. Absolutely no product inquiries or any questions will be taken on these lines.

INTERNATIONAL PHONE NUMBERS: Canada 1-800-548-2512; Italy 1678-74086; United Kingdom 0800-89-1178; France 19-0590-1099; W. Germany 01-30-810466; Japan 0031-11-1351; Australia 0014-800-125-712; Netherlands 06-022-8613; Norway 050-12029; Denmark 0434-0297; Switzerland 046-05-3420.

INTERNATIONAL ORDERING POLICIES: We have operators versed in the following languages: English, French, and Spanish, only. Shipping prices vary and are quoted following your order. Payment is made via credit card or electronic fund transfer. For any other information please call or regular toll lines. Absolutely no shipping inquiries will be accepted over the toll-free phone lines.

WE CAN BEAT ANY ADVERTISED PRICE! BUT WE SELDOM HAVE TO — OUR PRICES ARE THE LOWEST! (IF YOU SEE A LOWER PRICE ON AN ITEM, GIVE US A CALL.)

Amiga Dispatches

...And now, the **right** way to install FFS on Janus

by Don Curtis

Copyright © 1989 by Donald A. Curtis

Don Curtis is a police officer in Denver, Colorado, who has for the last three years been assigned to program design and development along with system design and maintenance of ten AT & T Unix 3B2 computers. In his spare time, Don is an assistant sysop on CompuServe's Amiga forums.

As you may have noticed by now, I gave you a bit of misinformation last time. I listed a method whereby you could use the new Fast Filing System (FFS) on an MS-DOS side Hard Disk partition for the Amiga (JANUS drive). As I stated in that article, I hadn't tried every procedure discussed, but felt that they were all from reliable sources. In this particular case, the information came directly from one of the software folks at Commodore. To put it simply, he was wrong.

I have a Bridge-equipped A2000, and do have a hard drive (10 Meg) on the MS-DOS side but, since I also have a hard drive on the Amiga side, I don't need to use a JANUS drive. Besides that, I need all 10 Meg for the MS-DOS side. Well, to make a long story short... someone told me that he had tried to use FFS on a JANUS hard drive and the drive wouldn't format. The symptom reported was that the format program would start fine, but would simply stop after formatting the first cylinder and hang. Nothing - no error messages, no activity, just a locked program.

I figured something had to be wrong in his mountlist or the procedure used, so I made a backup of my MS-DOS drive and tried to run FFS on a JANUS drive myself. Same problem: the format started, but hung. I tested, changed parameters in the mountlist, and used a sector editor to try to see what was actually occurring on the drive. I must have run FDISK and ADISK and rebooted, mounted and so on thirty times one night trying to get it to work. I knew the procedure I gave was close - the right type of activity was going on in the drive and the first cylinder was really being formatted, but nothing more.

The one quirk I did notice was that the normal format program, DPFormat, always started at cylinder 0 rather than cylinder 105, which was the actual cylinder where the JANUS partition started. Of course, DJMount and DPFormat worked fine, but that wasn't what I wanted.

That was clue number one: while I could access the drive if I MOUNTed it, my sector editor didn't show the proper information as to drive size and number of sectors. After playing around with the sector editor and mountlist entries, I learned that as far as the Amiga is concerned the JANUS drive is a completely separate drive, not just a partition on a drive. Changing the cylinder numbers in my mountlist to match those given by DPFormat when it runs took care of that problem. Now, if I used DJMount to mount the drive and DPFormat to format it, I could just use the normal Amiga MOUNT command rather than DJMount and I could still use the drive. Unfortunately, that didn't help with the Format problem. It still just hung.

Clue number two was the way the file system identifies a disk. The difference between the Old File System (OFS) and FFS is the way data sectors are used. OFS only uses 488 bytes of the 512 available for data; the other 24 bytes are file system information, mainly for recoverability. FFS gives up that recoverability but uses all 512 bytes for data. However, the root, directory and file headers are identical for both file systems. AmigaDOS uses a 'marker' in sector 0 of the file system to tell it what kind of disk you have. A KickStart disk uses the ASCII characters 'KICK' as its identifier. An OFS disk uses the ASCII characters 'DOS' plus a byte holding the value 0 (DOS0) as its identifier. An FFS disk uses the ASCII characters 'DOS' plus a byte holding the value 1 (DOS1) as its identifier. Remember, that's not the string 'DOS0' and 'DOS1', but the string 'DOS' followed by the value 0 or 1.

Now that my mountlist was right, if I mounted the drive as an FFS drive, I got a 'Not a DOS disk' requester. If I mounted the drive as an OFS drive, it worked fine. Both file systems recognized the disk. The format program didn't work using either OFS or FFS but, once formatted, OFS worked fine.

What would happen if I changed the identifier to the FFS identifier? Could FFS then use the disk? It sure can! I took my handy dandy sector editor and changed the 4th byte from value 0 to value 1 and wrote it out. I then rebooted, MOUNTed the drive with an FFS mountlist, and used it under FFS! By the way, AmigaDOS apparently uses only those first four bytes, so if you have a sector editor that requires you to checksum the block before you can write it, it will work anyway.

That was the final step. It's a kludge, but it works. You've got several steps to go through, but it's worth it in the end. The only format program that seems to work on a JANUS disk is DPFormat. DPFormat only understands OFS. FFS only understands a DOS1 type of disk, but OFS and FFS are enough alike that, in this case anyway, formatting under OFS and changing the disk type to DOS1 disk allows it to be used with FFS. Just to make sure, I ran a few tests:

Diskperf shows the following results for OFS:

File create/delete : create 9 files/sec, delete 16 files/sec
Directory scan : 51 entries/sec
Seek/read test : 49 seek/reads per sec

Read and write speeds in bytes/second:

buf 512 bytes, rd 20641, wr 17476
buf 4096 bytes, rd 20480, wr 17133
buf 8192 bytes, rd 20164, wr 16487
buf 32768 bytes, rd 20321, wr 16591

Not very impressive, and notice that higher buffer sizes made no difference. Now, the same thing under FFS:

File create/delete : create 8 files/sec, delete 12 files/sec
Directory scan : 106 entries/sec
Seek/read test : 57 seek/reads per sec

Read and write speeds in bytes/second:

buf 512 bytes, rd 23405, wr 21845
buf 4096 bytes, rd 68985, wr 55775
buf 8192 bytes, rd 68985, wr 56987
buf 32768 bytes, rd 77101, wr 59578

Quite a difference! And this was with an older 10 Meg drive that tests out at an average seek time of 95ms on the MS-DOS side. If you have a faster drive (most are), I suspect you'll get even better numbers.

Now, how about a real world test? I did a copy of all the files from my Amiga side hard disk onto the JANUS drive using the COPY command with the ALL option. Since this was only a 5 Meg partition, obviously it wouldn't hold all the files, so I timed it from the start until the JANUS drive filled. Using OFS, this took 8 minutes and 40 seconds. Using FFS, it took 6 minutes and 30 seconds to fill the drive and, since FFS uses all 512 data bytes per block, it also copied an additional 32 files (5% more) onto the drive.

Loading and running programs show that FFS on the JANUS drive was significantly faster than the OFS. I ran a few more tests to make sure that neither the Amiga partition nor the MS-DOS partition on the drive had problems with each other. They didn't. Sector editors on both partitions showed all the data just where it was supposed to be. I did find out one thing, though - do not try to use the MS-DOS side during JANUS disk activity. You will bring it to its knees! MS-DOS just doesn't understand doing two things at once, something I do quite often on the Amiga side. If you had Transformer, the software MS-DOS emulation program, you know how slow it is. Well, trying to use MS-DOS while accessing the JANUS drive is about ten times slower yet!

So... FFS does work on a JANUS drive, but not the way I said it should in my last article. Here's my method (there may be others - I just didn't discover them):

1. Back up everything on your MS-DOS side and also on the Amiga side if you already have a JANUS partition.

2. *Unless* you already have JANUS partition, you must now partition the MS-DOS side drive using the MS-DOS FDISK and ADISK commands (don't forget to have FDISK make its partition active). You'll have to reboot the whole Amiga after using FDISK and ADISK, so this step takes a bit of time.

3. After the Amiga is booted, use DJMount to mount the Amiga partition.

4. Use DPFormat to format the Amiga partition. While DPFormat is running, write down the low and high cylinder numbers! This is important. You *must* format the drive. Even if you already had a JANUS drive set up under OFS, you must reformat it now.

5. With a sector editor, go to block 0 of the drive. You'll see the DOS0 identifier as the first 4 bytes. Change byte 4 from 00 to 01.

6. Now create a mountlist for your new drive. You can call it anything you want, including JH0:. Use the example mountlist from page A-3 of the Enhancer Manual, but change the *Device* entry to **jdisk.device**. Change the *Unit* number to 0 for the first partition. Change *Surfaces* and *BlocksPerTrack* to match your drive. Change *LowCyl* to match the low number given by DPFormat, and *HighCyl* to match the high cylinder number. Here's what my mountlist looked like:

```
JH0:
   Device = jdisk.device
   Unit = 0
   Flags = 0
   Surfaces = 4
   BlocksPerTrack = 17
   Reserved = 2
   Interleave = 0  /* keep this at 0 */
   LowCyl = 0
   HighCyl = 153
   Buffers = 20
   Stacksize = 4000
   GlobVec = 1
   FileSystem = L:FastFileSystem
   DosType = 0x444F5301 /* Note this is 'DOS1' in hex */
   #
```

7. Remove any commands in your startup-sequence that refer to the JANUS drive, including any DJMount command. You won't use them anymore.

8. After saving your mountlist and new startup-sequence(s), wait the customary 3 or 4 seconds, then reboot the Amiga.

9. After your startup runs, MOUNT your FFS JANUS drive using the MOUNT command that came with v1.3.

10. Test your drive - see if you can access it. If you did everything right, it should work fine. If it does, you can now go back, restore any files you used to have on the drive, then re-edit your startup-sequence to **MOUNT JH0:** (or whatever you called it).

If you can't access the drive, go back over the steps above. Make sure you follow them exactly. This procedure worked fine for me and several others who tried it after I figured it out. I did not try using more than one partition for the Amiga, but I see no reason why it wouldn't work with multiple JANUS partitions. Just do the same thing with each partition (change the DOS0 to DOS1 after DPFormatting the partition) and set up a mountlist for it. The UNIT numbers would go up for each partition: the first partition is Unit 0, the second is Unit 1 and so forth.

Comments?

I have mixed feelings on the current state of affairs when it comes to the amount and type of software available for the Amiga. This also includes my thoughts about the A500 because the two ideas are tied together. Certainly, the A500 is a 'good thing' right now. It's the most popular of the Amiga line, and is selling well. In fact, it may be the machine that 'saved the bacon' for Commodore. The A1000 was selling, but nothing like the A500. The A2000 in its various incarnations is also selling well, but again, not as well as the A500.

It's because of the A500 that sales of the Amiga are now in the one million range. That's good. My concern is not for today, but for the future. In the long run, the A500 may not be a 'good thing'.

The A500 has opened up the Amiga line to folks who simply enjoy an excellent game machine at a reasonable price. That's fine - those folks deserve the best darn machine they can get for their money, and the Amiga seems to fill this niche quite well. My fear is that it may fill it too well.

The A500, while expandable, isn't nearly as convenient to expand as the A2000 and its descendants. The A500 becomes cumbersome when you add to it. A box sitting on the side of the system unit/keyboard combination works, but just doesn't feel right. You can't just shove the keyboard to get into a more convenient typing position. You can't pick it up and put it on your lap while you put your feet up on the desk (my favourite typing position while reading messages on CompuServe).

That doesn't make it any less of a computer, it just means that it's not as convenient a computer for certain things. In the long run, it also means that the majority of A500s will stay pretty much with their basic configuration... with perhaps a second floppy and a bit more memory. That also means that A500 owners won't be purchasing software that requires several megabytes of memory and/or large hard drives.

Of course, there will be exceptions to this. Some folks will boost their A500s to 8 Meg of memory and add a 100 Meg hard drive. But that certainly won't be the typical A500 owner. The typical A500 owner will use it for minor word processing, small home databases, some video work, and a lot of games.

To sell to businesses, we need high quality software, and don't need the reputation of a game machine. While selling to a business may not seem like a big deal, it is. Take my office for example. Even though I'm currently assigned to our data bureau, where we all do various levels of program development, there are more computers in the office than there are in our homes. I'm not talking about the main computers, I'm talking about the desktop computers. All eleven of us have computers on our desktops, but only seven of us have computers at home (two of *those* are Amigas!). Like most businesses, the machines we use are MS-DOS computers. It's because the software we need is available and the machines aren't known as a game machine. They're certainly not the best computers for the job, but we know that they won't be 'here today and gone tomorrow'. It's that simple.

I've owned computers in the past that gained the reputation of a 'game machine'. Unfortunately that meant that, after a while, only game software was available for it. The so-called serious software dried up for those machines. Oh sure, there were a few word processors available, along with some home-style databases and money-management programs, but there was little or no top quality 'serious' software - just tons and tons of games.

A lot of folks would say, "So what!" to that. They have no need for serious software. They couldn't care less about selling to businesses. They're perfectly happy with the available software. But there are also those who see large gaps in the software available for the Amiga. I'm not into video work, but I certainly see the need for software for those who are. Likewise, I'm not into games. I play a few, but generally don't have time for them. That doesn't mean others don't want more and better games.

That's where the A500 comes in. Publishers know the demographics, they know that if the large majority of units sold are lower-priced models, with limited expandability... that entertainment software will sell better than other forms of software.

But we need productivity software also. In fact, what we really need is a good balance of software between entertainment and productivity. That is what will keep sales of the Amiga healthy over the long haul.

Thumbing through recent issues of Amiga magazines, I see an awful lot of full page ads for entertainment software, and only a few pages of ads for productivity software. The mail order houses show about the same thing. The balance isn't there, but it's close enough for right now.

All I'm saying is that I'd like to see it stay close to balance. The Amiga is a very versatile machine. It can play great games, and it can also run fantastic software... but the software has to be there. I'm asking publishers and developers to consider what type of software they offer. Some of the so-called productivity software I've seen on the Amiga is plain and simple garbage. It's pushed off as professional-level software, yet it's cumbersome, slow and inflexible.

What good is DataBaseX (boy... I hope there's not a product by that name!) if it can't at least import a dBASE III file? I'm not saying it needs to handle dBASE programs. But no business is going to be convinced DataBaseX is a professional level program if it can't accept their data without having to re-enter 30,000 records by hand.

Word processors that take an hour to print a letter because they print *all* the letter in super hi-res graphics just aren't going to cut it. That may be fine for an occasional letter from home, but a business that puts out 10 or 20 letters a day can't wait that long. They care about properly-formatted letters, on letterhead with clean, crisp text. They have the money to buy a quality printer that doesn't need to run in graphics mode to get the crisp text. I'm not saying that graphics output should not be provided for those who need it... but don't ignore the ASCII output for those who can use it. Allow the users to select their preferences; don't force your preferences on them.

If you're going to take the time to write a program, take the time to do it right. An extra month spent polishing the program can make the difference between night and day. Listen to the users, find out what they need and provide it for them. Look at what you do, what you need to get your job done... and try to provide the software to make that easier for you, and everyone else that does a similar job.

If you're writing a game, great... but make it the best darn game you've ever seen. Then take some of the tricks you needed to get the game to run and apply them to an application program.

Don't get stuck in a rut, vary your product line. In the long run, you, me (the user) and the Amiga will all be winners!

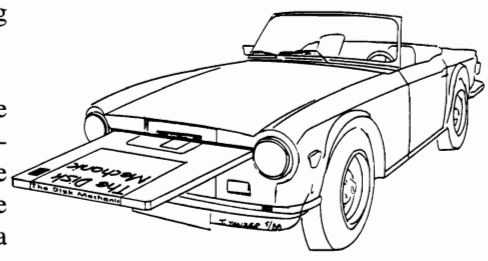
Just because the A500 is the most popular machine in the Amiga line, doesn't mean it's the only Amiga worth considering. Don't ignore it, but don't write only for it. If the software is there, the larger machines will pick up sales also. The Amiga is a great machine to play a game on, but it is *not* a game machine. It is capable of doing things most could only dream of a few short years ago. Let's continue to take advantage of those capabilities in every way possible.

. . . if you have at least 2 years of 6800 family assembly language programming experience, or have an NTSC/PAL Video Hardware background. Join the team that makes no compromises and help us change the world. Key positions available at the fastest growing Amiga company. Hackers call Marla Svoboda at 913-354-1146 or write to Marla at 115 West Crane Street, Topeka KS 66603.

Tired of waiting? Give your disks a Tune-up with The Disk Mechanic

Only $89.95

The Disk Mechanic is a comprehensive collection of Amiga DOS utilities for every application. **The Disk Mechanic** can recover files that have been deleted, salvage files from corrupted disks, and repair damaged files. **The Disk Mechanic** includes a disk optimizing program that can increase your hard or floppy disk access speed up to 400% by reorganizing the disk's data. **The Disk Mechanic** also includes a hard disk back-up program and a full featured disk block editor for the advanced user. Version 2.0 of **The Disk Mechanic** includes full support for the soon to be released Amiga Fast Filing System and a new high speed hard disk back-up utility. **The Disk Mechanic** requires an Amiga with at least 512K of memory and Amiga DOS version 1.2 or higher. Call us or ask your dealer about it today!



Lake Forest Logic Inc.
28101E Ballard Road
Lake Forest, IL 60045
(312) 816-6666

Dealer inquiries welcome.
Amiga is a trademark of Commodore Business Machines, Inc.

MasterCard VISA

Mastering The Raster

The professional video niche

by Derek Grime

Derek Grime is president of Beyond Graphics, a Toronto-based studio that uses the Amiga for a variety of graphics, animation and video applications. In this new column, Derek talks about using the Amiga professionally, the Desktop Video market, and other concerns in his field. Derek can be contacted at Beyond Graphics, P.O. Box 1249, Station F, Toronto, Ontario, Canada, M4Y 2V8

I purchased my Amiga 1000 in October 1985. It had 256K of ram, one internal drive, no monitor and a hopeless prerelease version of the operating system. I was between jobs and for me it was ridiculously expensive. My first box of disks cost almost seventy-five dollars.

When I got it home I played with the clock. I made big clocks and little clocks. I set the time. I set the alarm. I opened and closed them. I dragged them around. I calculated how many big clocks would crash the machine. Then I investigated how many small clocks would crash the machine. I showed my Amiga to everyone I knew. They made strange twirling motions beside their heads. I had a hell of a good time.

My dream was to take my training in animation, illustration and high end computer graphics and apply it to the Amiga. We know this now as desktop video but before the Amiga there was none. This computer even had a video-out jack on it. The hardware was in place but software was a problem. The clocks had little commercial promise.

A dealer took pity on me and gave me a demo version of a paint program. At the time there was nothing like it anywhere. Graphicraft. Three years ago it was a revelation. Within two weeks I had done my first desktop video job. In hindsight it was horrible, but I was hooked.

Since then I've purchased more Amigas and hired other artists. We've produced many network television jobs with our Amigas. We animated Commodore's Canadian television commercials for the past two years. We were CBM's first VAR (Value-Added Retailer) and we've helped make the Amiga the DTV system of choice for advertising agencies, video production studios and educators in Ontario. We work with many of the very best hardware and software developers creating the new products that push the envelope of Amiga graphics. My studio likes to do the things that you're not supposed to be able to do on the Amiga.

That's what this column is all about. Pushing the envelope of Amiga desktop production as well as taking our beloved box to the lofty heights that it deserves. We're passionate about the Amiga and its future. Passion is not just great affection but can also include outrage and infuriation. When you rely on a computer for your livelihood the relationship runs both hot and cold. I'll be examining both sides every month.

Holes in the marketplace

So let's start on a positive note. The Amiga is maturing as a visual production system, but still there are many much-needed tools that haven't reached the market. Some special graphics applications may never be released by the big guys because the numbers just aren't there. During consulting sessions with the larger developers I've mentioned many of these holes in the Amiga marketplace and some of this software could even be in production. Then again, "real soon now" means very little to all us Amigans. Since *TransAmi* is the magazine of choice for writers of quality software, fire up your favourite compiler and read on.

As boring as it would be to write, we need business graphing software. Pity the poor visionary who fought the powers that be to get an Amiga for his in-house media service department. Much of his time is spent hand-drawing charts and graphs that could be created in seconds by software. Imagine a program that you feed numbers and titles to that generates those 3D pie, bar and scatter graphs. Aegis tried it once upon a time with Images but today's user requires something a little more sophisticated. Programs like this abound in the IBM world and

you know we could do better. In fact, if it produced vector files as well as IFF pictures, it could easily be adapted to drive film recorders that give thousands of lines of resolution on 35mm slides. While charts have little sex appeal, they do raise the pulse rate of users in the business market. Until now business users have been virtually ignored. They have lots of money to spend on the right software. What are you waiting for?

Three dimensional graphics on the Amiga are big business as well. While I'll save a comprehensive wish list for another column, there is a dire need for more object creation programs. File incompatibilities have been addressed by John Foust's *Interchange,* God bless him, but the creation programs themselves still leave much to be desired. Each one seems to have one drop dead feature. That means the artist must jump from one to another, Interchanging all the way. A good modeller is not a light programming chore. A good interface is by far the most important feature here. Take some inspiration from the high end systems. Visit a computer graphics trade show and see what I'm talking about. A modeller should also be able to read all formats from outside the Amiga community and should write files for them as an option. Plotter support is important here as well. Industrial design is another Amiga software orphan.

Every video job that is produced on the Amiga is storyboarded. Storyboards are like comic strips that the director and creative staff use to guide a production. Not only do they show a rough graphic but they contain information on the type of shoot, scene number, dialogue and sound effects. Microfiche Filer can be twisted to do the job but I'm thinking more along the lines of a unified package. A production manager would like to have all the budget and billing information close at hand. Critical path analysis is terrific to make sure that all the pieces are under control. Include a text editor for scripting and invoicing. Take those individual storyboard frames and sequence them at the proper speed so the client can approve shots and timing at every step. Include sound samples and an SMUS player. Macintosh users have simpler variations on this theme and I personally know of several studios that do the video on the Amiga and everything else on their Macs. Something is wrong here.

Colour cards are on the way that will let us display thousands or millions of colours on the Amiga. New paint programs are needed to support them. It goes without saying that they should be selectable between the standard graphics modes and the card's limitations. Any resolution should be selectable, even though some finesse is required to get around the chip RAM limitations. Overscan has to be there of course, in normal, severe and ridiculous levels. Colour registers should be fluid, allowing the artist to reset the palette at any time. Bit planes need to be increased, decreased or turned off if necessary. Ham and half-brite can't be ignored. Differing palettes should remap as an option when they are loaded in. Configuration files should load at start-up and contain palettes, brushes, fonts and other options. The paint program market is still wide open. Deluxe Paint II remains the reigning queen of graphics software but the old girl is showing her age. What software is going to capture the single biggest sales category on the Amiga?

Let's talk about the blitter. The other guys now have the same screen resolution and colours on screen that we've grown accustomed to. Of course it costs three times more on the other machines but who's counting? Our most important advantage is the blitter. The Amiga draws pixels faster than a bat out of hell. In many cases the game developers exploit this more than the graphics programmers. We need new video software.

After DPaint the one program that we use in more television jobs than any other is still Deluxe Video. Why do we use something that my peers consider marginally better than a toy? Because it gets the job done. Over two years old, this work horse allows only a total of sixteen colours in low resolution mode. There is no ham, half-brite, severe overscan, pretty file requesters or other niceties. Screen rewrites are painfully slow. However, the interface is a lovely thing to behold and the resulting animations are smooth. Many more recent attempts have been made to move bobs in hi-res and the results are mostly a jerky disappointment. In the studio we play with the Director but without the interface we can't make it pay its way on a job. Gold Disk's Moviesetter adds the interface but loses the resolution. Solving the graphics catch 22 will make someone very wealthy and the users very happy.

Please don't forget the little things that users hold near and dear to their hearts. Give them toys they'll miss if they ever try your competition. File requesters should sort by variable options and patterns, list dates and file sizes, and show only available devices. Don't add colours to a requester if it is going to use any more RAM or slow anything down. Here are some pet peeves I hear again and again. Every function should have a keyboard equivalent - remappable, if possible. The mouse is a great input device but I can still move my hands faster. Give people from the Workbench and the CLI camps a choice. Let's see more ARexx support. Faster chip sets can't be ignored any longer. The first paint program that looks for the 68020-30 will be a huge success. Support multitasking. Support hard drives. Drop title screens. Drop copy protection. In the next few months I'll be addressing many of these points in this column. Stay tuned.

The Amiga family has been with us for over three years now. Worldwide sales are said to be right around the one million mark. These are numbers that the competition can't ignore any longer. Possibly the only thing holding back the Amiga from a real graphics renaissance is the lack of solid, well designed video/art application programs. The professional user is often forced to look back to the past for solutions when they should be looking to the future. These holes have to be filled. As an Amiga developer, imagine the joy you'll bring to others by helping them create better and faster than they ever have before. Imagine the joy you will bring to your wallet. If you use one of these ideas I can only humbly request that you allow me the opportunity to crash your software before it reaches the rest of the world. The marketplace awaits.

Assembly Programming, Part 1

Building a program from scratch

by Jim Butterfield
Copyright © 1989 Jim Butterfield

This is the first part of a series of articles on Amiga assembly language programming by Jim Butterfield. Jim's many articles, books, lectures and television programs have made his name a household word among Commodore users throughout the world since the early days of the KIM-1 microcomputer.

Introduction

It seems to me that assembly language programming on the Amiga is hidden from the average beginning programmer's view. It's hidden in many ways: in pre-written 'include' code, in macros, and in external definitions.

That's hard to avoid when you get to big programs. But here's what it means to the beginner: the program you see published is often a small fragment of the whole job.

When you become an expert, you too will take the short cuts. That annoying and finicky start-up code will indeed be set aside and linked in as 'canned' material. You'll automatically know the most popular three dozen external definitions and happily call them by name. You will put together favourite code chunks of your own: for example, code to output a decimal number will be needed over and over again. Write it once - well - and link it in as you need it.

But as a beginner, you'd feel you were walking into the middle of a novel without having read the first few chapters. It's hard to feel comfortable that way.

So I'll try to tread slowly in these early sessions. Later, you'll find some of these beginning things tedious, and rightly stash it away as part of your library.

Absolute beginners?

It's reassuring to get to the job of useful coding as quickly as possible. To make it there, I'll have to gloss over some areas where an absolute beginner will require further study. Such things as: binary and hexadecimal notation; the address and data buses; the cycles of fetch-an-instruction and then execute-an-instruction; input/output devices; and the operating system itself - all these will be galloped by as if they were perfectly obvious.

Forgive me if I go too fast on these. There are books in which you can gather some of these concepts, and supplementary reading is a valuable learning tool.

The basic tools

You will need an *assembler*. There are both public domain and commercial packages available. The first assembler program offered by Commodore was from MetaComco, and this still works well. C compilers may include an assembler as part of the package. And there are several quite good 'freely distributable' assemblers available: one by Wes Howe, which has a few small bugs; and another by Charlie Gibbs which, in its latest 1989 version, is full-featured and quite bug-free.

Either of these should be available, free, from your local user group or bulletin board. Two 'published' assemblers - AssemPro by Abacus and *COMPUTE!*'s ASM68010 - are available; these will do the job, although they may be somewhat limiting for big programs.

About 'include' files: These usually come as part of the assembler package. Don't worry about them too much at the moment, but you'll be glad to have a fine set of these before you go too far. As we develop, you'll see the ones you need.

You will need a *linker*. This will almost certainly be BLINK, the splendid package by John Toebes and others of The Software Distillery. The programs you write will be assembled into *object files*; these cannot be run until they are *linked* and made into *executable* files. At first it will seem like an unnecessary extra step, but you'll learn to appreciate it later.

You'll need a *debugger*. This is the same thing that was called a *Machine Language Monitor* on the 8-bit Commodore machines. WACK was the original Commodore debug program; it was soon followed by the commercial program MetaScope. The Abacus AssemPro system has a debug program as part of the package; and an excellent public domain debugger, still

under revision, has been produced by Jim Thibodeau and Larry LaPlume.

You'll need *reference books*. For starters, buy only one Amiga reference: The *AmigaDos Manual* (Bantam Computer Books). Although on first inspection the book seems to be just a CLI guide, you'll soon discover that it contains a section entitled *AmigaDOS Developer's Manual* that tells you almost all you will need for your beginning exploration. The best 68000 reference material comes from the manufacturer of the chip. *The MC68000 User's Manual* (MC68000-M/AD) is published by Motorola, and you may also find the "Family Reference Guide" (MC68000FR68K-D) of value. In North America, they may be ordered from LVC Phoenix (800-521-6274); in the Toronto area, they are stocked by Electrosonic.

Personal bias

You'll notice that I keep my nose very close to the code at the start, using no include files or external definitions. I'll grow out of that soon; but when I do, you'll be called upon to look more closely at the assembler program you've got. There are wide variations between packages, and I can't write code to suit every wrinkle. You'll have to decide what your package needs.

I will be staying clear of customized Amiga features for a while. Such things as graphics and sound will come later. My first emphasis will be on 'text flow'. You'll start by reading and writing to and from the screen and disk files.

It seems to me that text is a good place to start. It will allow you to ask for data, send messages, and do a great deal of useful stuff. But if you're hypnotized with graphics, sounds, blitters and coppers, you won't read about them here for quite a while.

Background bits

Memory is organized in *bytes*, as with most microcomputers you might have encountered. The 68000 chip can happily read and store individual bytes from its memory, using any appropriate address. But there's a more powerful way to access memory: two bytes at a time.

Two adjacent bytes may be brought in or stored in a single memory action, so long as the lower byte is at an even address. This two-byte pair is called a *word*; it's a very efficient way to use memory data. You may reference both bytes in a single instruction by supplying the address of the lower, even-numbered byte. As an example, you can obtain (or store) the contents of addresses 20 and 21 in a single action. But you cannot reference the contents of 21 and 22 in a single shot; if your program tried this, the computer would guru with an 'address error'.

You may also reference four bytes at a time; this is called a *long word*. The computer can't handle a long word in memory in a single action; it will have to take two dips to do the job. Again, the lowest address is the one you use, and it must be an even value. Thus, a long word reference to address 4 would actually handle the four bytes at addresses 4, 5, 6 and 7.

Figure 1 shows the three data configurations, plus one more: addresses. These usually occupy a long word. Since 68000 addresses are only 24 bits (three bytes) in size, the high order part of the long word is not used, and should be set to zero.

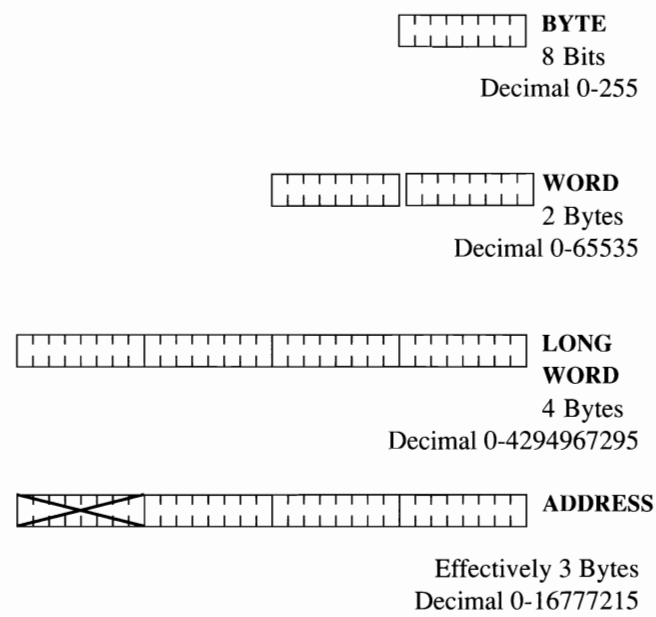


Figure 1: 68000 Data types

68000 Registers

Inside every microprocessor you'll find storage and calculation areas called *registers*; there are quite a few of them in the 68000. We'll briefly mention two special ones, and then move on to the main working registers. The PC register is the *Program Counter*; it tells you where the computer is working at this instant. The SR is the *Status Register*, which holds information (called 'flags') on the condition of the computer and its data. We'll mention these again later, but for the time being we'll move on to the main action area: the data and address registers.

There are eight *Data Registers*, designated D0 to D7. You may use them in several modes: byte, word, or long word. If you use a data register in a byte mode, for example, only its lowest byte will be used, changed, or tested. Word mode uses only the lower two bytes of the register. Long word, as you might expect, uses the whole thing.

You are free to use any data register for any purpose you like. It's handy to know that register D0 is most often the one that receives data back from a subroutine that you might call; for that reason, you might like to keep D0 relatively free. And register D0 and D1 might be changed by a call to a system subroutine, so it's best not to keep long-term data in these.

There are eight *Address Registers*, designated A0 to A7. You usually put the whole long word to work in these registers, so you don't play around with byte mode in these. A word mode is possible; but if you use it, the word value will expand to fill the whole address register.

Register A7 is special. First, it's a 'double' register: there's a second copy of A7 that you'll never see, or need to worry about, until you get into high powered stuff like 'supervisor mode'. Second, it's the main stack pointer for your program. You may use it, but treat it with respect. If you leave it alone, it will take care of itself very nicely.

Register A6 has a special meaning on the Amiga. It needs to be used as the 'library pointer' any time you call an Amiga library - and you'll do this, lots of times. Reserve A6 for this use; we'll deal with it very soon.

The other address registers, A0 through A5, are free to be used any way you like. Keep in mind that the contents of A0 and A1 may be changed by system subroutines, so you may want to keep these two fairly clear.

If you have programmed on the Commodore 8-bit computers, you might find it useful to think of the address registers as doing the same job as zero page on the 6502. We will certainly need them when we get to indirect addressing (but not this time).

As you start to explore the built-in Amiga libraries, you'll discover that they like you to set up subroutine calling data in the lower-numbered address and data registers. So you'll tend to reach for the higher-numbered registers to hold your longer-term data and addresses; that way, you'll have less need to juggle the information.

A first program

Gosh, all this theory and no hands-on yet. Let's set up a project and get to the coding. It seems to me that the most popular beginner's program (in other languages) is one to print "Hello, World!".

This program will run from CLI only, to save us the complexities that a Workbench program involves. We'll call our program *Test*. So when the command *Test* is typed in the CLI or Shell, the program should cause "Hello, World!" to be printed on the screen.

We'll examine what needs to be done in a bottom-up manner, starting with the job we need to do. To print a message we will use the DOS library command *Write*.

The write stuff

The form of the *Write* command is as follows:

```
returnedLength = Write(file, buffer, length)
   D0                  D1    D2     D3
```

D1 - file = file handle
D2 - buffer = pointer to text buffer
D3 - length = integer, length of text
D0 - returnedLength = integer, confirmed length of text.

Location: -$30 in the DOS library jump table.

You'll learn to read this type of format fairly quickly. Going through the information, we see that buffer and length seem fairly straightforward: that's our message. The returnedLength value confirms that we did the output successfully. But what in the world is a file handle?

A *file handle* is an identifier that tells the system what input or output stream should be used. That's all very well, but how do we get this number? It's not as simple as the 'logical file' number that you may have dealt with on the Commodore 64.

How do we get the file handle? If we opened a file, we'd get it automatically. But we need to get the file handle of our existing output path. No problem: we use *Output*.

The form of *Output* is:

```
file = Output()
D0
```

D0 - file = file handle
Location: -$3C in the DOS library jump table.

The *Write* command called for us to supply three pieces of information. *Output* requires no values; we just call and receive our file handle value in D0.

Our programming task now seems clear. We call *Output* to get the file handle of our output stream (the CLI window, unless it has been redirected); using that handle, plus the address of our "Hello, World!" message and its length, we call *Write*, and out it goes.

But there's a hitch. The two routines we need - *Output* and *Write* - are located at -$3C and -$30 in the DOS library jump table. But where the devil is that jump table?

Libraries and jump tables

The DOS library isn't at any fixed place in RAM. It will likely be in a different place in your computer than in mine. In principle, it might not even be in RAM at all, in which case it will need to be brought in. In principle again, it might even have moved since the last time we used it.

So we must find the library, bringing it in if need be, and then nail it down so that it won't move during the time that we need it. And we do all this with a single subroutine: *OpenLibrary*. Later, we must release the library when we don't need it any more; we do that with *CloseLibrary*.

Here's the form for OpenLibrary:

```
LibPtr = OpenLibrary(LibName,Version)
D0                   A1      D0
```

A1 - LibName = pointer to Library name string
D0 - Version = version number required, zero for any
D0 - LibPtr = pointer to library, zero if not found.
Location: -$228 in the Exec library jump table.

So to find the pointer to the DOS library base, we set up a string called "dos.library", neatly followed by a zero byte to signal the end. We put the address of this string in A1, a zero in D0, and then we call the subroutine *OpenLibrary.* When it returns, it will contain either zero, meaning the library was not found (you must have spelt its name wrong), or the address of the DOS library, which is what we want.

You may have detected an apparent paradox here: how do we find the Exec library so that we can call *OpenLibrary* and *CloseLibrary*? Trust me: we'll get there with no trouble. Stay with the central flow for the moment.

Once we have found the pointer to the DOS library, we can easily find the entries in the jump table for *Output* and *Write* by using the offsets from that pointer. The offsets are negative values; we count *down* from the pointer to get to the appropriate jump table entry. Thus, if the DOS library pointer shows address $C04C70, our subroutine jump to *Output* would be hexadecimal 3C bytes *lower*, or to address $C04C34. Don't worry about the numbers too much; the computer will do the arithmetic for you. Depending on the debug package you use, you may find it handy to know that negative values are held as two's complement numbers by the computer, so that -$3C might look like $FFFFFFC4 when you put your debugger to use.

Taking stock

You might find it useful to see our program as it is currently outlined.

1. Find the DOS library by calling *OpenLibrary.*
2. Get the output handle by calling *Output.*
3. Send the message by calling *Write.*
4. Release the DOS library by calling *CloseLibrary.*

This last we haven't quite documented, so here's the form.

```
CloseLibrary(libPtr)
             A1
```

A1 - libPtr = pointer to library you wish to close.
Location: -$19E in the Exec library.

The only missing link is: How do we find the Exec library?

Answer: Easy. Its address is always in the long word starting at address 4.

Into the code

First, a quick note that we need to put two strings into memory. One is the DOS library name ("dos.library") followed by a binary zero. The other is our message. We'll preview these two lines of code; they will not really be needed until the end of our program.

```
; DosName  dc.b  'dos.library',0
; Message  dc.b  'Hello, World!',$0a
```

The labels "DosName" and "Message" mark the addresses at which the strings start. DC.B stands for "define constant, byte", and means: here come a bunch of bytes to be placed in storage. The bytes can be characters, such as "dos.library" (be careful of upper and lower case here), or values such as the binary zero that terminates the first string. The second string does not need a terminating zero; we'll count the characters and tell *Write* how many to send.

Don't put these lines into your program yet. I'm showing them so that you'll know they are coming.

Invading Exec

The address of the Exec library is stored at address 4 (long word). When we use an Amiga library, we first put the library pointer into register A6. So let's do it.

```
MOVE.L $4,A6
```

Move, long word, the contents of address 4 into register A6. Now the Exec pointer will be in A6. Before we call *OpenLibrary,* we must set up a pointer to the library name in A1. A good way to get an address into a register is to use the LEA (Load Effective Address) command. We might type **LEA DosName,A1** and that would work. But we'll be more specific and say:

```
LEA DosName(PC),A1
```

The PC notation means this: work out the address by calculating its *offset* from this part of the program. It will be fifty bytes or so ahead, but the computer will work that part out. Specifying PC will save us a bit of memory, although the program would work however it happened to figure out the address of the string at DosName.

Now for the version. We'll take any DOS we can find, so a zero will do. We can put a zero into register D0 in several possible ways. For example **CLR.L D0** (clear long word) would do the trick. I'll pick the MOVEQ (move quick) command.

```
MOVEQ #0,D0
```

Now we're ready for the call to the library function. Register A6 contains the pointer to the library. **JSR (A6)** would call a subroutine (Jump Subroutine) at that address, but that would

be folly - we'd miss the jump table completely! Note the use of parentheses to signal an indirect address - we jump not to A6, but to the address contained in A6. But we don't want to jump there; rather, we wish to use an offset of -$228 from that address to do an *OpenLibrary*. So here goes:

```
JSR -$228(A6)
```

This will call *OpenLibrary*, an entry in the Exec library. But for heaven's sakes, don't forget A6, which points to the library. Leave it off accidentally, and you'll jump into oblivion. The system may crash so fast it won't even see a guru on the way out.

When the *OpenLibrary* routine returns, register D0 should contain a pointer to the DOS library. If it contains zero, it means the library failed to open: you've probably spelt the library name wrong, or forgotten the binary zero at the end. Let's check in any case, as we move the library pointer to A6:

```
MOVE.L D0,A6
```

The address we obtained is now in A6. But what if it's zero? In that case, we had better skip the rest of the job. After this last instruction, the Z (zero) flag will be set if we have moved all zeroes. So we can check with a BEQ (branch equal) instruction. I know that this will be a short hop, so I'll identify it as a 'short branch':

```
BEQ.S Exit
```

Branch Equal (to zero) short, to "Exit", a line we'll write in a few moments. At this point, we'll make the rest of our job a subroutine, and call it with a BSR (branch subroutine). Again, this will be a short one:

```
BSR.S Main
```

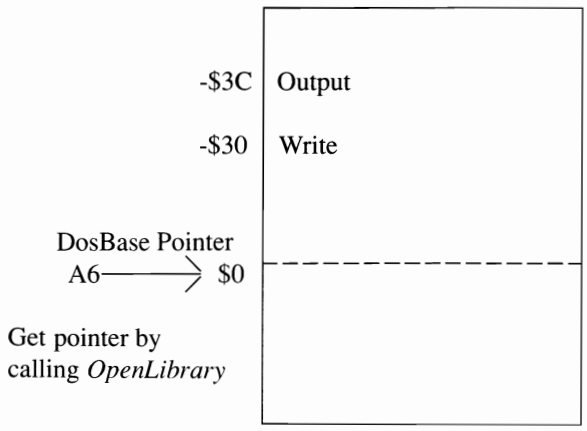


Figure 2: The DOS library jump table, somewhere in memory.

Why did we call this subroutine with BSR (Branch), when we made a previous call using JSR (Jump)? It's a matter of closeness. We know that subroutine Main will be nearby, so the more compact Branch instruction will save us memory. The system calls can't use BSR for several reasons, the simplest of which is that the library is probably too far away to reach with a Branch.

When our main job is complete, we'll come back here, close the DOS library, and quit. Our DOS library pointer should still be in A6. You may recall that *CloseLibrary* wants to see this pointer in A1, so:

```
MOVE.L A6,A1
```

Let's get the pointer to the Exec library back into A6 now:

```
MOVE.L $4,A6
```

We're ready to call *CloseLibrary*, located in Exec at an offset of -$19E:

```
   JSR -$19E(A6)
Exit:
   RTS
```

The setup job is done. Now we're ready to apply ourselves to the task we set ourselves.

Brief interlude

The above code sets things up so that we can go about our main work. Similar jobs would use exactly the same code to get things started. This kind of logic is called 'start-up code'. We use it (or a more ornate version) over and over for almost everything we do. After a while, it gets boring. So we might do any of several things: put it in a file and bring it in with an 'include'; make it into a 'macro', a piece of canned code; or assemble it and bring it in with the linker when it's wanted.

After you've written a few dozen programs, you won't want or need to look at start-up code in most cases. You will dive straight into the main job, and assume that the canned start-up code is in order.

We'll track this code the first few times, however. Once in a while, it's nice to be able to dig into this area and customize it to your needs.

The main job

The start-up code calls this as a subroutine (with a BSR). A6 contains the address of the DOS library pointer. We're ready to go. First, our call to *Output* to get our file handle. No data set-up is needed:

```
Main:
   JSR -$3C(A6)
```

The handle is now in D0. We'll need it in D1 for the *Write* operation:

```
MOVE.L D0,D1
```

Now to bring in the address of "Hello, World!" and put it into D2. The LEA (load effective Address) command works only with address registers, so we'll do it in two steps:

```
LEA  Message(PC),A0
MOVE.L A0,D2
```

We'll count the number of characters in the message manually, this time. Including the exclamation point and the newline character (hex A), I get 14 characters, so we move quick with:

```
MOVEQ  #14,D3
JSR  -$30(A6)
```

Our call has been made to *Write*, and our task is done. Return back to the start-up code with:

```
RTS
```

Finally, here are our two strings, mentioned before:

```
DosName  dc.b  'dos.library',0
Message  dc.b  'Hello, World!',$0a
```

That's the whole job. Let's do a repeat of the whole program, with a few 'equates' to make things more readable, and one or two comments.

Most assemblers call for labels (sometimes called symbols) to be flush with the left margin; lines without labels should be indented. That convention is followed in the listing below. Comment lines, which start with a semicolon character, vary according to the assembler; I've indented them for greater readability.

```
   ; Hello program
   ; Startup sequence, CLI only

   ; Equates for Exec Library
_LVOOpenLibrary     equ   -$228
_LVOCloseLibrary    equ   -$19E
   ; LVO means Library Vector Offset

   MOVE.L $4,A6              ; Exec base
   LEA    DosName(PC),A1     ; Dos name string pointer
   MOVEQ  #0,D0              ; any version
   JSR    _LVOOpenLibrary(A6)
   MOVE.L D0,A6              ; Dos base pointer
   BEQ.S  Exit               ; Zero, quit
   BSR.S  Main               ; Do the job
   MOVE.L A6,A1              ; Dos base pointer
   MOVE.L $4,A6              ; Exec base pointer
   JSR    _LVOCloseLibrary(A6)
Exit:
   RTS

Main:
   ; Main Job
   ; Equates for Dos library
_LVOOutput          equ   -$3C
_LVOWrite           equ   -$30
   ;
   JSR    _LVOOutput(A6)     ; Get output handle
   MOVE.L D0,D1              ; Handle to D1
   LEA    Message(PC),A0     ; Address of Message
   MOVE.L A0,D2              ;  .. to D2
   MOVEQ  #14,D3             ; Length of Message
   JSR    _LVOWrite(A6)      ; Send message
   RTS

DosName  dc.b  'dos.library',0
Message  dc.b  'Hello, World!',$0a

   end
```

Putting it together

Using your favourite editor, type it all in and save it, using the name "TEST.ASM". Be very careful about upper and lower case when using labels and library names. If you have the Amiga.lib file that comes with developer's kits or commercial assemblers, you could change the *equ* lines above into something more convenient. You would use an *xref* (external reference) here and ask the linker to find the proper offset values. We'll pick up on that next time. For the moment, the above program is complete in itself and needs no external support.

You may now ask the assembler to go to work. The usual way is to type **<assembler-name> TEST**, but check your assembler's documentation in case it needs something different.

If you get errors, go back, fix them, and try again. Eventually, you'll get a clean assembly, which produces an *object* file that can usually be recognized by its *.o* or *.obj* suffix. In other words, you should now have a file called TEST.O or TEST.OBJ - but your program is not yet ready to run.

To get an executable program, we must 'link' TEST.O so that it becomes TEST. Our program is unusual, by Amiga standards, in that it's complete in itself. In other words, it doesn't have to be linked to anything. But we must still run the linker program before it will be ready to go. A few assemblers, such as AssemPro, skip the linking step and generate an executable program directly; if this is the case, you won't need to link. Command **BLINK TEST.O TO TEST**, or **BLINK TEST.OBJ TO TEST**, and your program will be converted to an executable form, ready to run.

Type **TEST** and the computer should reply, "Hello, World!". You've put together your first simple Assembly Language Amiga program. Feel good?

Implementing An ARexx Interface

Design decisions, problems and solutions

by Charles B. Blish

Charles Blish is president and founder of Black Belt Software, a Montana-based Amiga software development company. Among the Amiga products he has developed are the CAD programs PCLO and PCLOplus, formerly marketed by SoftCircuits Inc.

By now, most developers will be aware of ARexx - and if not, they certainly should be. Why? Because you can save yourself a *lot* of work while adding a great deal of power to your product.

In brief, ARexx provides three basic capabilities to a product. First, you can have macro capability. Second, this can be extended to a full externally-driven script capability. And third, you get an inter-process communications port (IPC) in the bargain.

The body of this article is aimed at those considering the inclusion of an ARexx capability of some level in their product. I'm assuming some familiarity with ARexx from a user's point of view - I think most of you should have that at this point. If not, then you *really* need to take a look at ARexx. Not only for your end users, but for yourself... It is no exaggeration to say that ARexx is one of the most important time saving system additions you can buy.

Levels of support

There are three levels at which you can approach the inclusion of ARexx in your product. You can do a 'minimum' implementation, which has an ARexx message port, and understands a message from ARexx of the form 'CLOSE'. ARexx can always launch your program (essentially via the **execute** capability in DOS) and with the addition of a CLOSE command, it can also cause it to terminate. This isn't hard to do, and probably should be required in every product. You can also go 'all the way'. Here, you implement every blessed thing that ARexx makes possible in your product. This would include the following capabilities, and possibly others:

- Macros
- Scripts (external and internal)
- Multiple, unique IPC's
- Macro/Script re-entrancy
- Macro/script recursion
- Process-local script editing

You can also implement something in between these two extremes. You have to look quite closely at your product in order to determine which capabilities you need, and and how extensive the implementation will be. When the ARexx interface for *PCLOplus* was created, we decided to do a 'full boat' job on it. The reason for this is fairly straightforward: We saw ARexx as a means to change our product in a major way, for a relatively minor effort.

Decisions, decisions

PCLOplus is a CAD program for designing printed circuit boards. Previous to the ARexx additions, it was completely interactive - that is, the user did something with mouse, menu, or keyboard, and the program did something in return. No more, no less. Although we felt that *PCLOplus* was a good program, the addition of a smart script language (and ARexx is a *very* smart script language) brought *PCLOplus* directly into the realm of the big boys - we could do full **autorouting**, a process whereby a printed circuit board is constructed by the program, according to specific rules and using the trace specifications provided by the user - and all this without more than a few weeks work on our part.

When you design an ARexx interface, you need to ask yourself the following questions: Do you need local macros (such as FKEYs the user can define on the fly, while running your program)? Do we need recursion and/or re-entrancy? Do we want more than one copy of our program to run at one time; and if so, do the other copies get their own ARexx IPC, do they share the first IPC, or do they run completely without an IPC? Of course, you can simply let them have the same IPC name, but that's sloppy.

You get script capability no matter what, as soon as you add even one ARexx command, so that's not really an issue. Only the extent of the scripts is, by which I mean the number of commands your program recognizes and the amount of parameter handling it can do.

Some implementations simply provide a command structure that is identical to operations already available in the program. For instance, let's say that the program uses menus, as any

good major Amiga application does. Often, within the program, you will have one procedure that you call when you get a menu event from Intuition. Here, it's very convenient to build a set of command language keywords that are translated directly into menu messages, identical with those Intuition would have sent you had the user actually selected that item in the menu. Once this is done, you simply call the procedure with the message. At this point, you have a substantial beginning on a script language. You can continue this approach by providing every key stroke, mouse click, and gadget selection that the user can do while running the program, and you have a complete ARexx script language built. Additionally, and this is important, you are using the knowledge that the user already has of the program so that learning this new capability will be relatively painless. Relatively?

Yes. Because ARexx is not without its pitfalls for the user. ARexx has a syntax that is both powerful and primitive at one time - and this extends to the IPC details that you must implement.

ARexx's communications back to the ARexx script that is running are extremely weak - it's difficult to provide a detailed response to a script in a generalized way beyond one string of data - and even that isn't all that easy. ARexx has but one variable that is used by every call to the host program. It must constantly be shuffled around by the user's script, which can be very frustrating for the script-writer. When you create your ARexx interface, you'll have to design around this somehow, and making a script interface with poor feedback isn't a whole lot of fun.

ARexx and accessibility

Next, you can't just implement ARexx and expect it to be used by your average Amiga user. It's just not possible. It's a programmer's language, not easily learned and not easily programmed in, either. While Bill Hawes has provided a very good *reference* manual, it's missing something very important: a tutorial section. Without that, new users will mainly be in one of two classes. Either they have used it before (on the IBM or it's clone brethren) or they are programmers who love the new, unknown, and potentially powerful. So here you have this neat new capability in your program, and the number of people who can actually *use* it is minimal. This is not a good situation, and hopefully will be rectified.

Another problem is demonstration software. Currently, you cannot demo your script capabilities, because Mr. Hawes has not provided a version of ARexx that you can ship with a demo. There is one exception to this (*TxED* from Microsmiths); but this was a one-time situation, and there has been no progress on a demo version despite continuous prodding from us, if no one else. This is a real mistake, viewed from where I sit. In our case, a PCB CAD program that autoroutes a circuit board after getting some parameters from the user is a much better demo than a PCB CAD program that comes with a notice that says, "read the docs." This will apply to you as well. Editors that show off fancy capabilities, graphics packages that draw, fill, and so forth - all upon boot-up - make much better demos than a static package does. For now, you'll have to live with it, though. Of course, you *can* pester Mr. Hawes about it...

The speed problem

One more issue is ARexx's speed - or lack of it. ARexx is painfully slow; especially when viewed from the perspective of a programmer type who is used to compiled languages, having discarded BASIC and its ilk long ago.

Immediately, you see a problem here: With a primary user group who are programmers, this becomes a major issue. To be fair, it's not anyone's fault, really. ARexx uses the Amiga's message system to pass information and commands back and forth with your application and this means that there is a very definite upper limit on the amount of message traffic that can occur within a given time period. You need to be aware of this *before* you start your implementation, so that you can create some very powerful commands locally (in your application). This will bring the speed of an executing ARexx script back up to some degree and will make the use of the language much smoother for this particular audience.

One last point on this: ARexx is an interpreted language, and so isn't particularly fast in its own right. That fact combined with the IPC bottleneck adds up to the noticeable speed problem.

Lest you think I'm on Bill's case, it's not so. He has done a fine job in implementing an Amiga version of Rexx, and he is tops when it comes to direct (telephone) support of people implementing ARexx interfaces. The reason that I can brag that *PCLOplus* has one of the most sophisticated ARexx interfaces to date is as much due to Bill's support as to any efforts of mine.

I do think that ARexx may be the single most important application support product to come along since *Power Windows* (you *do* know about *Power Windows* from Inovatronics, don't you?). And unlike *Power Windows*, which simply helps you (a *lot*) while you are designing a product, ARexx is there after you release your application. And the ARexx users are there, enhancing your program's capabilities long after you've gone on to something else.

Finishing touches

Until an object-oriented (Iconic) analog of Rexx comes along, ARexx is 'it' when it comes to IPC and script capability. An object-oriented version, with which a 'normal' user could point and click his way through to a complete script or macro, is what I perceive off in the mists somewhere.

And, speaking of icons... You need to look at the Workbench side of things, too - tooltype options, 'project' icons for external scripts, and so forth. Keep in mind that although ARexx is a text-oriented tool, Workbench users often have valid needs

in this area that you should address. One thing I have done for this type of operation, which you may wish to consider, is to include a local edit capability within an application. In brief, this is simply a special-purpose ARexx script that invokes the user's choice of editor with one of the user's macros as the *parameter* for the editor. Let's say that the user has ten ARexx Fkeys available, and Fkey 2 contains the string **DoStuff.rexx**. I set it up so that pressing the Control key and the Fkey at the same time would invoke the user's editor like this:

```
UEDIT DoStuff.rexx
```

It's trivial to do, and it removes the need to constantly be switching back and forth between a CLI and the particular application. These are the kind of touches that the user will appreciate. If you take your time and really think the ARexx implementation through, you'll have done everyone concerned a big favour - you, and the end users.

In conclusion, I would say that you should, if at all possible, include ARexx in your application; and that the implementation should be as complete a set of your application's capabilities as is practical. Those who ignore ARexx do so at their own peril. Simply scanning the ads in any of the Amiga magazines will reveal the words "ARexx Interface" a good percentage of the time now. Once a person has ARexx and realizes what it can do for him (in whatever environment he first encounters it), he is certain to begin pondering what it can do in other applications. If your competition has it and you don't, I think you'll lose the sale. It's as simple as that.

Bill Hawes deserves a hearty pat on the back for providing such an important tool to application developers.

Oops...

The previous issue of *TransAmi* (Volume 1, Issue 5), had a mix-up right on the cover: Anthony Bryant's article about using Interface Technologies' M2AMIGA Modula-2 system was announced as "Using TDI's M2AMIGA". TDI's Modula-2 system is used in Clark Williams' article in the same issue, which may help to explain the confusion, but is obviously no excuse for the error. The article on M2AMIGA did not mention Interface Technologies either, which was another oversight. For those of you left wondering about the source of the M2AMIGA system (which Anthony was quite enthusiastic about in his article), it is from: Interface Technologies, 3336 Richmond, Suite 323, Houston, TX 77098-9990. Phone (713) 523-8422

The fastest
Modula-2
Software Development System for
AMIGA
$ 249
Demonstration disk $ 10

Modula-2 is the successor of Pascal. Both languages were invented by Professor Niklaus Wirth. Modula-2 is an all purpose language suited for educational, home and professional use. Modula-2 leads to good programming technique, easy readability and maintainable programs. In fact, a lot of schools and universities have chosen Modula-2 for their programming courses.
M2Amiga is a software development system for Modula-2 based on an extremly fast single pass compiler. It is fully integrated into Amiga Workbench and is easy to use from CLI or Workbench. It runs on all Amiga models with a minimum configuration of 512k RAM and one disk drive.
M2Amiga has been developed specially for the Amiga and therefore optimally supports the features of this unique computer.
- Single pass compiler produces optimized native MC68000 code from standard ASCII files. Full support of latest edition of Modula-2.
- Fastest library calls possible on the Amiga. No intermediate code needed for library calls and parameter passing.
- Supports FFP, 32 bit IEEE and 64 bit IEEE reals formats, you can even mix all three formats within the same program.
- Easy access to MC68000 registers, allows the insertion of inline code.
- Full interface definition to the Amiga's operating system following the official definitions.
- The best runtime system currently available on the Amiga. Opens and closes all used libraries, captures all GURUs and releases all used resources after program crash or termination. Routines for System requesters enable user interface consistent with the Amiga's Intuition.
- The emacs based editor places the cursor on compilation errors and explains them in full text.
- In just a few seconds the linker produces stand-alone applications that can be run from CLI or Workbench. Stand-alone applications do not contain any useless overhead – they are compact and fast.
- All programs can produce icons for their output files to allow effective use of the intuitive Workbench environment. Experience the new feeling of program development by selecting and double clicking icons!
- Also included are a project management tool and other utilities.
- Besides the handy manual, an introductory guide to Modula-2 is included. This documentation allows both the beginner and the advanced user to find any information needed quickly and easily.
- Demo programs show the use of various features of the Amiga.
- For professional program development special utilities are available.
  + Source Level Debugger, the new way of debugging your programs. Inspect all your variables (displayed in their own format), zoom into structures as records or arrays or follow pointer chains by clicking the mouse buttons
  + Object file converter, converts standard Amiga object files into M2Amiga format. This allows linking of routines written in other languages.
  + Linker for devices or libraries. Add power to the Amiga by supplying new libraries and devices. Easy and intuitive programming instead of fooling around with Assembler and Amiga linker are possible now!
  + Modula-2 Amiga Programming System Environment

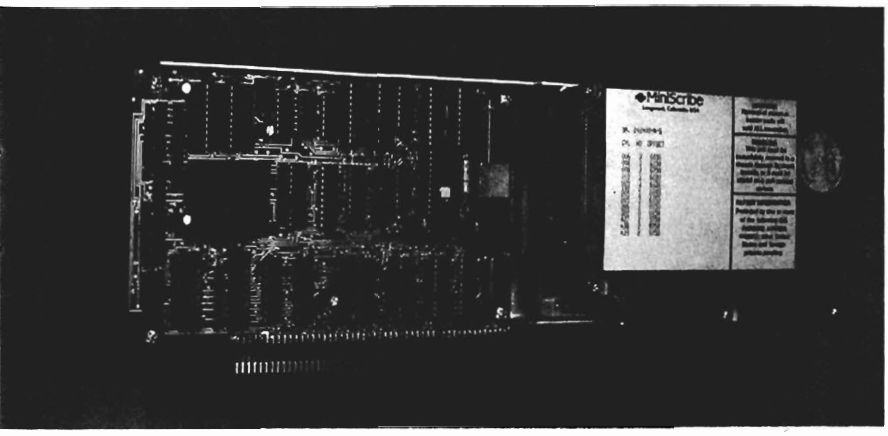
We have Modula-2 Compilers for HP-UX, IBM/370, PCs (Taylor, M2SDS, JPI), OS-9 and Sun, tomorrow even more!
Texas residents add 8% sales tax. International Orders add $ 20 shipping.

The Modula-2 people:
INTERFACE TECHNOLOGIES
3336 Richmond, Suite 323, Houston, TX 77098-9990 (713) 523 8422
Dealer inquiries welcome

A. + L. Meier-Vogt
Im Späten 23
CH-8906 Bonstetten/ZH
Switzerland
Tel. (41) (1) 700 30 37

HardFrame/2000

The *Super-Speed*, DMA, SCSI Hard Disk Interface for the Amiga®2000



- **AutoBoots AmigaDOS 1.3 *(Price Includes HardFrame Eprom!)***
- ***Directly* Boots the New Fast-File System! *(Doesn't Need Old FS!)***
- **Auto-mounts All Hard Disk Partitions *(no Mount List Required!)***
- **Designed-in, Ultra Strong, Multitasking Performance**
- **High Quality Metal Frame for Stable, On-Card, Hard Disk Mounting**
- **Power Cabling Directly from Card to Disk**
- **50-pin Cable Included**
- **Supports up to *seven* SCSI hard disks of any size**

How fast is *fast*? **HardFrame/2000** transfers data at Amiga bus speeds! It's actually faster than the hard disk mechanism itself! And even more important in the Amiga's multitasking environment, **HardFrame/2000** has extremely efficient DMA circuitry to get on and off the bus in almost no time at all: 280ns to get on; 200ns to get off. And it's true, dedicated DMA, too! **HardFrame/2000** autoboots *and* automounts *directly* into the AmigaDOS™ 1.3 Fast File System (old file system partitions are *not* needed!). The core of any DMA SCSI interface is in its SCSI protocol chip and DMA chip. MicroBotics has chosen the new, high performance Adaptec AIC-6250 SCSI chip, capable of up to 5 megabytes per second raw transfer speed, and the Signetics 68430 DMA chip running at 12.5 megahertz. Then we added additional FIFO buffering and enabled 16-bit wide data transfers for maximum throughput. The sophisticated design of **HardFrame/2000** provides for automatic SCSI arbitration, selection and reselection. The hardware supports either synchronous or asynchronous data transfer. **HardFrame/2000** can function as either the SCSI bus initiator or the target and can reside in a multiple master environment. Physically, **HardFrame/2000** is optimally flexible: the compact, half-size card comes attached to a full length, plated aluminum frame. The frame has mounting holes positioned to accept standard, 3.5" SCSI hard disk units such as those manufactured by MiniScribe, Seagate, Rodime, and others (hard disk mechanisms must be supplied by the user or his dealer as a separate purchase item). Alternatively, you can cable-connect to a SCSI drive mounted in your Amiga's disk bay or in an external chassis. As many as seven hard disks may be connected to a single **HardFrame/2000**. There is no size limit on each disk. **HardFrame/2000** includes a 50-pin SCSI cable and header connectors for either 50-pin or 25-pin cable connection. Also included is a current tap to power frame-mounted drives directly from the slot itself. **HardFrame/2000** comes complete with driver, installation, and diagnostic software.

Available NOW! Suggested list price, $329 (hard disk not included)
Frameless version: $299.00. See your Amiga Dealer.

The HardFrame/2000 photo shows the product with a MiniScribe twenty megabyte hard disk installed. Hard disks are *not* included in the purchase price of HardFrame. Note that if placed in the first slot, HardFrame uses only one slot even with a disk attached.

New!
8-UP! (DIP) FastRAM

Another great memory board from MicroBotics, 8-UP! (DIP) is the "brother" of the original 8-UP! (which uses SIMMs and PopSIMMs to fill its memory space). 8-UP! (DIP) uses conventional 1 megabit RAM chips in standard sockets to provide your Amiga 2000 with 2, 4, 6, or 8 megabytes of autoconfiguring FastRAM! 8-UP! (DIP) is a super efficient CMOS design for lowpower consumption and high reliability. Suggested list price, $199 (0k installed)

Join MicroBotics ONLINE TECHNICAL SUPPORT CONFERENCE ON BIX (The Byte Information Exchange) –call 1-800-227-2983 for BIX membership information!

MicroBotics, Inc.
Great Products Since the Amiga Was Born!
811 Alpha Drive, Suite 335, Richardson, Texas 75081 (214) 437-5330

Tell your dealer he can quick-order from MicroBotics directly – no minimum quantity –show him this ad!

"Amiga" is a registered trademark of Commodore-Amiga. "HardFrame/2000", "8-UP!", "PopSimm", are trademarks of MicroBotics, Inc.

Memory Allocation On The Amiga

How fragmentation chokes the system

by Eric Giguère

Resource allocation is one of the most important functions of an operating system. It takes on even greater importance in a multitasking system such as the Amiga since multitasking involves allocating the CPU between several tasks or processes. Memory is also a resource that must be shared among the tasks, both when programs are loaded and when they are executing. This article is concerned with *dynamic memory allocation* - memory requests made by programs in execution.

An example

Many C programs will request blocks of memory from the system at one time or another. The program accompanying this article is one such example. It isn't particularly interesting - all it does is build a huge linked list and then cuts it down by half. And by so doing, it will probably render your Amiga useless. Type it in and try it out - please remember to save whatever you were working on before running it and reboot when you're done.

For all its simplicity, the program touches on three issues related to memory allocation:

1. *Proper allocation of memory* - Are your memory requests too small? Too large?

2. *Freeing memory* - When your program stops, does it return all the memory it requested back to the system? In what order is the memory freed?

3. *Handling low-memory conditions* - If the system cannot fulfil a memory request, will your program bomb?

To address these issues and figure out why the program does what it does, we first have to look into some background material.

Memory and the Amiga

Memory allocation on the Amiga is controlled by Exec, the lowest level of the Amiga operating system structure. Chapter 6 of the *Amiga ROM Kernel Reference Manual: Exec* describes the various system routines available to the programmer for memory allocation. The ones used most often by programmers are AllocMem(), to allocate a block of memory, and FreeMem(), to return a previously-allocated block to the system. AllocMem() has the function prototype:

```
void *AllocMem( long size, long flags );
```

where **size** is the amount of memory (in bytes) desired and **flags** chooses various memory options (found in the header file *exec/memory.h*>:

```
MEMF_CHIP - allocate from CHIP RAM
MEMF_FAST - allocate from FAST RAM
MEMF_CLEAR - clear memory before allocating
```

(Each option corresponds to setting a bit in the flags parameter and so they can be ORed together - passing **MEMF_CHIP | MEMF_FAST** would allow the system to allocate either from CHIP or FAST RAM.) A pointer to the start of the allocated memory block is returned. AllocMem() returns NULL if it cannot fulfil your request.

FreeMem also takes two arguments:

```
void FreeMem( void *ptr, long size );
```

where **ptr** points to the start of a block that was previously allocated with AllocMem() and **size** is the block length.

A typical memory allocation/deallocation sequence might go like this:

```
char *ptr;
ptr = (char *) AllocMem( 100L, MEMF_CHIP|MEMF_CLEAR );
/* do some stuff...*/
FreeMem( (void *) ptr, 100L );
```

Two important points to remember:

1. *Never* try to free memory that wasn't allocated by AllocMem()

2. *Never* use memory that's been freed by FreeMem()

Both might cause a system crash. The second point is extremely important because it's a common error to re-use memory that's been freed. If the system has allocated it to some other process, you'll likely be stomping over that process' data.

Technical details

Exec has to keep track of what memory is available before it can dole it out. It does so by maintaining a circular linked list called the *free list*. The free list consists of all contiguous blocks of memory not currently allocated to any process (or in use by the system). When a memory request is made, Exec scans the free list for the first available block that's large enough to satisfy the request. If it's too large, Exec splits it up and returns the leftover portion to the free list. When a process frees a block, Exec simply inserts it back into the free list.

The linked list is stored in the free memory itself. The first four bytes of a free block contain a pointer to the next free block; a second set of four bytes contains the size of that free block. This works because AllocMem() always allocates memory in multiples of eight bytes, so the minimum size of any block is eight bytes. If your memory requests are small in size, you may be wasting as much memory as you are allocating. (By the way, these numbers aren't guaranteed to hold in future revisions. I'm presenting them purely for interest's sake. Don't depend on them.)

Fragmentation of memory occurs when a lot of small blocks of free memory are separated by blocks of allocated memory.

Exec tries to *coalesce* adjacent blocks of free memory whenever it can. Larger blocks make it easier to satisfy subsequent allocation requests. (This is all the housekeeping that Exec can do; moving free blocks together is an impossible task, since the blocks of allocated memory in between cannot be moved.)

Rules of memory allocation

1. *Try to keep the size of memory requests to a minimum.*

2. *Always check to see if your request has been satisfied.* It's better to exit gracefully (freeing all your resources) than to just crash when you run out of memory. Some programs can be written to automatically adjust to suit the available memory.

3. *Don't hang on to memory.* If you're done with it, return it to the system. Other programs might need it.

4. *Free all the memory you allocated before your program terminates.* Exec loses track of memory once it's been allocated - it only knows what's been explicitly freed. Even program termination isn't enough to return any allocated memory to the free list.

5. *If you can, use FAST memory before CHIP memory.* CHIP memory is an expensive commodity best saved for graphic and audio purposes. (Of course, not all Amigas have FAST memory, so use the combination **MEMF_FAST | MEMF_CHIP.** FAST memory is checked first.)

The program

By now you should be able to figure out what the accompanying program does. It's an extreme example of memory fragmentation. First, it allocates all the memory it can into a doubly-linked list. It then proceeds to free every second element in that linked list. When this is done, it tries to allocate a block of memory slightly larger than the size of a list element. It can't, however, because of fragmentation; all the free blocks are too small. On my Amiga 2000 the program told me I had one megabyte of FAST memory free; yet it couldn't allocate a block of nine bytes! Not only that, but the program doesn't free all its memory upon termination, leaving the Amiga with a permanent case of fragmentation. That's why you should reboot, especially if you run it using CHIP memory.

Granted, this example is rather contrived; the linked list elements are exactly eight bytes in size and the program assumes that memory allocation is performed in a linear fashion. But it is a good demonstration of what can happen in the worst situation.

Final notes

Memory allocation isn't particularly complex when compared to other facets of the Amiga operating system, but it is crucial to the operation of the machine. Use the Exec routines properly and consistently to avoid a lot of heartache.

More information on memory allocation schemes can be found in Donald Knuth's *The Art of Computer Programming, Volume I: Fundamental Algorithms*, published by Addison-Wesley, and in most textbooks on operating systems.

"frag.c" - Memory fragmentation at its worst.

```
/*
 * frag.c -- Memory fragmentation example by Eric Giguere
 *
 * ================================================
 *
 * WARNING! WARNING! WARNING! WARNING! WARNING! WARNING!
 *
 *   This program may cause your system to crash!!!
 *                    Use carefully!
 *                  Reboot when done!
 *
 * ================================================
 *
 * Compiles under Manx C 16-bit option, should
 * work with the 32-bit option and on Lattice C
 *
 */
```

```
#include <stdio.h>
#include <exec/memory.h>

/* Choose which type of memory you wish to access */

#define MEMORY_TYPE   MEMF_CHIP

#define putstr(s) fputs(s,stdout);

/*
 * Define our linked list structure
 */

typedef struct list
  {
    struct list *prev, *next;
  }
    list;

/*
 * Now declare the memory functions
 */

#ifndef AZTEC_C  /* Lattice C prototypes */

  static void  prt_long( long );

  extern long  AvailMem( long );
  extern void *AllocMem( long, long );
  extern void  FreeMem( void *, long );

#else /* pre-ANSI declarations */

  static void  prt_long();

  extern long  AvailMem();
  extern void *AllocMem();
  extern void  FreeMem();

#endif

/*
 * Now allocate memory and test for fragmentation
 */

void main()
  {
    list *head, *new, *old;
    long  size, avail;
    int   ch;

    putstr( "Fragmentation test:  Reboot your system when done.\n\n" );
    putstr( "Do you wish to proceed? (Y or N)\n" );

    while( ( ch = getc( stdin ) ) != 'y' && ch != 'n' )
        ;

    if( ch == 'n' )
        exit( 0 );

    size      = (long) sizeof( list );
    avail     = AvailMem( MEMORY_TYPE );

    /* Make sure some RAM exists */

    if( avail < size )
      {
        putstr( "\nNo memory available for the fragmentation test.\n" );
        exit( 0 );
      }

    putstr( "\nAvailable memory: " );
    prt_long( avail );

    /* Now create a linked list using all the RAM available */

    putstr( "Allocating list...\n" );

    head      = (list *) AllocMem( size, MEMORY_TYPE );
    head->next = NULL;

    while( ( new = (list *) AllocMem( size, MEMORY_TYPE ) ) != NULL )
      {
        head->prev = new;
        new->next  = head;
        head       = new;
      }

    head->prev = NULL;

    putstr( "Cannot allocate any more memory.  Available: " );
    prt_long( AvailMem( MEMORY_TYPE ) );

    putstr( "\nFreeing every second element...\n" );

    old = head->next;

    while( old != NULL && old->next != NULL )
      {
        new = old->next;

        new->prev       = old->prev;
        old->prev->next = new;
        FreeMem( (void *) old, size );
        old             = new->next;
      }

    if( ( new = AllocMem( size + 1L, MEMORY_TYPE ) ) == NULL )
      {
        putstr( "Could not allocate element of size ");
        prt_long( size + 1L );
        putstr( "Memory available: " );
        prt_long( AvailMem( MEMORY_TYPE ) );
      }

    exit( 0 );
  }

/*
 * prt_long -- Print a long integer.  Use instead of printf.
 *
 * Assumes that '0' to '9' are contiguous in the character table.
 */

static void prt_long( val )
  long val;
  {
    static int buf[ 10 ];
    int i, ch;

    i = 0;

    while( val > 9 )
      {
        buf[ i++ ] = (int) (val % 10);
        val /= 10;
      }

    buf[ i ] = (int) val;

    while( i >= 0 )
        putc( buf[ i-- ] + '0', stdout );

    putc( '\n', stdout );
  }
```

68000 DISASSEMBLY

OTG Software

ANNOUNCING . . .

DSM VERSION 1.0d

DSM is a full-featured disassembler for the Amiga. Check out these features and you'll see why programmers agree, "DSM is the best disassembler currently available for the Amiga, bar none."

- DSM disassembles any Amiga program that does not use overlays.
- DSM produces assembler-ready output that is 100% compatible with the Amiga assembler, assem. Excellent results can also be obtained using the C.A.P.E. assembler from Inovatronics.
- DSM features true text detection.
- Improved disassembly is supported using the built-in expert system mode.

If your local Amiga dealer doesn't carry DSM, send check or money order to:

OTG Software
200 West 7th Street Suite 618
Fort Worth, TX 76102

TX residents add 7.25% sales tax.

Price
$67.50 US

TRY BEFORE YOU BUY!

Best selling games, utilities and classics, plus new releases!

- 100's of titles
- Low prices
- Same day shipping
- FREE brochure

YES, WE ACCEPT:

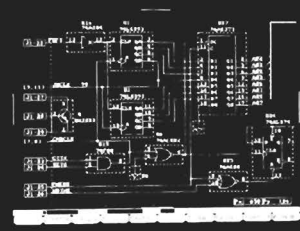
DISCOVER

RENT-A-DISC
Frederick Building #219
Huntington, WV 25701
(304) 529-3232

Prolific Inc.

Available Now!

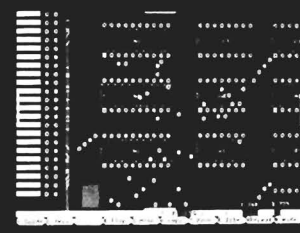
PRO-NET



$475.00

For AMIGA™ only

PRO-BOARD



$475.00

POWER+SIMPLICITY=PRODUCTIVITY

Feel our POWER. . .

PRO-NET:
- *Variable template size*
- *A to E paper size*
- *Extensive Library included*
- *Auto device number with Zone control*
- *Gate swapping*
- *Single click change to negative logic*
- *Innovative weight assignment*
- *Auto page reference*
- *Dynamic error checking*
- *Creates BOM, Spare Part List, Net List, Error report, etc.*
- *Supports printers & plotters*
- *Supports Laser Printer*
- *Back annotation from PCB layout*
- *Move, Rubber. . .and lots more*

PRO-BOARD:
- *.025 inch grid*
- *.001 inch grid Library*
- *12 mil trace, 13 mil space*
- *Produce 1, 2, 4 layer PCB*
- *Provides silk screen*
- *Auto coordinates assignment*
- *Auto produces power and grid layers*
- *Single line auto route*
- *Optional Net List Input for guided route, no need to look at schematic*
- *Prioritized route*
- *Dynamic error checking*
- *Supports printers, plotters and Gerber photo plotters*
- *Copy, Repeat. . .and lots more*

See our SIMPLICITY. . .

Intelligent Function Keys make our programs extremely user friendly, provide maximum screen area, always display all relevant commands, avoids excessive cursor movement and screen flashing between menu & drawing, guides user through operation, minimizes training time.

ORDER OR CALL FOR DETAILS
1808 W. Southgate Ave., Fullerton, CA 92633
Tel: (714) 447-8792 Telex: 5106016526 PROLIFIC CALIF
Western Union Easy Link Mail Box 62935949

TRY OUR DEMO DISKS FOR $15 EACH.

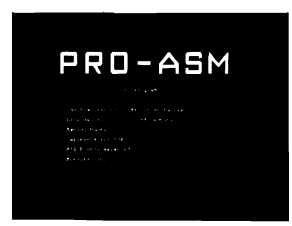
NEW!

Also from Prolific Inc.,
4 full feature AMIGA™ Macro Cross Assemblers for Z80, 6809, 8085 & 8051.

See our SIMPLICITY

— PRO-ASM —

Feel our POWER
- *Includes multi-pass Assembler, Linker, and Serial Down Load*
- *Generates relocatable Object Code Module*
- *Nested Macro*
- *Includes Files*



$85.00

Feel our POWER
- *Conditional Assembling*
- *Rich set of directives*
- *Global and External Variables*
- *Data format includes Binary, Motorola Hex, Intel Hex, and Tek Hex*

DEALER INQUIRY INVITED.

AMIGA™ trade mark of Commodore Inc.

BreakPoint Part Four

A look at Manx and Lattice's source debuggers

by Victor A. Wagner
Copyright © Victor A. Wagner, Jr.

Vic Wagner started his association with computers in 1965 while in the US Air Force working in a group studying digital flight simulation. Since becoming a civilian in '66, he has been working mostly with minicomputer manufacturers in the real-time systems software area. Weekdays from 8 to 5 find Vic answering the technical support line for Computer Automation Inc., and maintaining real-time software systems. Evenings and weekends he spends most of his waking hours talking to Taarna (his Amiga 1000), writing programs, and spending time on CompuServe's Amiga forums (76046,3004) where he is currently teaching a C class. Vic is a recognized authority on Metadigm Inc.'s well-known MetaScope debugging program.

Well it is *not* next month. Sheesh, it isn't even the next issue. A lot of things have changed since I last sat down at my Amiga to write an article on debugging. The biggest thing is that *Transactor for the Amiga* is now a monthly magazine (which means that I'm gonna have to write twice as fast). In the world of Amiga debugging there has been a new entry in the list of debuggers to evaluate (*CodePRobe*, from Lattice). Don't you just *love* those cute names that people come up with for products? *CodePRobe* with just the CPR capitalized. Well, since *CodePRobe* is the 'hottest' debugger out (well, it's the most recent anyway), I thought we ought to talk about it this month. Of course we'll talk about SDB also (Manx's source debugger) because they are currently the only two source debuggers available.

We will cover the remaining debuggers next issue (assuming my fingers can find their way to the keyboard again). I hear rumours of source debuggers coming from both Avant-Garde (*Benchmark Modula-2*) and Metadigm (*MetaScope* upgrade). Also, I'm attempting to find out the fate of *LDebug* (SoftCircuits) since they seem to have folded up... and somehow my copy of it seems to have disappeared.

At any rate, this month we'll look at the so-called 'source' debuggers, and what they mean to those of us who write in C. Although the Amiga object file format allows for debugging information to be inserted, and even though this information was published with the release of the Amiga, no 'standard' has been published on what this debugging information should look like. As a result, different techniques for getting information to the debugger have evolved.

If I may, I'd like to insert an editorial comment here: Part of the reason that the Amiga is as successful as it is surely belongs to the IFF standard. I find it disappointing that as time goes on, even that 'standard' is being eroded by new products and companies who *have* to do 'special' things. In the world of source debugging, I'm sorry to report, the information provided for debuggers is not even vaguely similar in format. Manx writes theirs in a separate file, Lattice buries it in the executable file (as allowed for in the original specification).

Unfortunately, each company seems to think that the information is proprietary. Certainly each company provides their users with a program which deciphers the information. Certainly I understand the desire to have the best product available, and I understand the desire to have an 'edge' on the competition. What I do *not* understand is this desire to have market share as a prime motivator. The introduction of strictly proprietary systems and software packages will lead to the disaster in the PC world (*lots* of programs, but no two will talk to each other). Bill Volk (formerly of Aegis, now of Mediagenics) summed it up best "I don't want a bigger piece of the pie. I want bigger pies". The way to bigger pies is for the companies to cooperate and share information, even if it is just their external data formats.

Okay, I'll get off my soapbox for a while, and we'll take a look at two remarkable programs. If anyone had told me (or the computing world at large) in 1966 that programs such as SDB and *CodePRobe* would be readily available, we would have locked him or her up in the looney bin. Of course, back then a *big* machine was 128K bytes. The debugging information required to keep track of a medium sized program - especially an Amiga program with all its system structures - exceeds 128K all by itself.

Let's look at some of the 'physical' aspects of the debuggers before we enmesh ourselves in the features of the programs themselves.

```
cpr          169392 ---arwed 07-Nov-88 15:13:40
sdb           89228 ----rwed 24-Jan-88 01:43:48
```

As you can see, *CodePRobe* is substantially larger as a file on the disk. In memory, on my machine, the usage looks like:

CodePRobe: address	size Hex	size Dec	SDB: address	size Hex	size Dec
0296F0	224	548	814F10	4	4
840660	50	80	8428A0	214	532
842640	1680	5760	87A188	2058	8280
87A188	7300	29440	87C1E8	76D4	30420
881490	6C7C	27772	8838C8	2B88	11144
888118	75B4	30132	886458	20AC	8364
88F6D8	7B70	31600	888510	2330	9008
897250	7D74	32116	88A848	3B54	15188
89EFD0	69D0	27088	88E3A8	26A0	9888
	=====	=====	890A50	2060	8288
Total	2D0D8	184536	892AB8	200C	8204
				=====	=====
			Total	1AB08	109320

Remember, that the Amiga does 'scatter loading'. The addresses are therefore not of particular interest, but the sizes of the hunks show that *CodePRobe* is also larger in memory.

One more physical difference is that *CodePRobe* comes with an impressive, imposing 6-1/2" x 9" 138-page manual with both index and table of contents. *SDB* comes with a 5-1/2" x 8-1/2" 66-page manual. Both manuals are well written and laid out nicely. Assuming that you've learned how to read manuals, you should have no difficulty in getting through either of them. The *SDB* manual has a section called "Tutorial" and the *CodePRobe* manual has a "Walkthrough" section. Both are useful reading if you're planning on using the program effectively. *SDB* comes with some excellent demos (scripts which walk you through some sample debugging sessions.

Folks, this is just one of those cases where reading the manual *first* is *highly* recommended. Although both programs have well thought out interfaces, I could not call either one 'intuitive'. Please don't take that comment as criticism of either: learning to use a program as complex as these is like learning a new language. The people who wrote the programs have different names for the same things, so the abbreviations and the commands differ slightly.

How they compare

Bringing up the programs, we notice differences right away. *SDB* has a special three-in-one window which opens up to 640x200. *CodePRobe* opens up two of its possible four windows and automatically sizes them to fill a screen that is the same size as your Workbench screen.

The three-in-one window is sizable to at least 680x432 and adjusts the sizes of its two internal 'windows' proportionally. You may be wondering why I call it a three-in-one if it has two 'internally sizable' windows. Well, one of the parts is one line tall and is where everything you type appears. The 'window' above is where the source appears, and the one below is where the responses to what you type appears. The ratio of the two windows is adjustable by clicking on a gadget at the right end of the command line and dragging it either up or down. The command line moves to the new position and the source and display windows expand and contract to fit. I like the idea of the command line for input; it doesn't take up space in the display window for each of the commands you've executed.

CodePRobe, on the other hand, uses a relatively straightforward dialog window. It resembles a CLI window quite a bit in that you see what you type at the bottom, and the window scrolls just like the ones with which you're familiar. Both programs have command line editing and history. One *big* difference I noticed was that SDB can *not* debug a program which has *not* been processed to have a *.dbg* file, whereas with *CodePRobe* you can at least get the disassembly of the program and debug at that level.

Debugging with these programs is a real joy compared to all the debuggers I used before I came to the Amiga. It is just simply amazing to me that I can type in a command to 'breakpoint' on line 65 of the current file and then look around at the variables (by name even) and decide what to do next.

Both programs have a bewildering array of ways to set breakpoints, put conditions on them (only break execution if some expression is true), execute them *n* times, etc. There are both permanent and temporary breakpoints as well. You can even tell the debugger to execute some commands when the break in execution occurs. The most likely commands are display/print commands, but the possibilities are endless.

Speaking of display/print functions, both of these programs will let you examine memory in almost any format you could possibly desire. For looking at raw memory, you can look at the data as bytes, words (Manx) or shorts (Lattice), longs, pointers, strings, floats... and the list goes on... and on... and on.

I think all those commands are just for us old fogeys who have been using them for decades. The real excitement comes when the program has been compiled with the requisite options so that the debugger can 'see' the type of the variables. Now all you have to do is tell the debugger to print a variable, and it shows up on the screen, in the correct format.

You can even display structures, and each of the elements gets displayed in its own correct format - the display even identifies the elements by *name*. *SDB* specifically states that it has a method to examine a function's parameters. I suspect that *CodePRobe* does also, but haven't run across it in the manual yet. If you're getting the feeling that I'm overwhelmed by these programs, you're catching on.

Let's take a look at a simple session with these programs, just to give you a feeling of what is possible. We'll assume that

you have successfully compiled and linked the program and have invoked the debugger. Under normal circumstances the debugger will come up with the source of your program in the upper window, and the next line to execute highlighted in some fashion. If you're into looking at the code the compiler generated (I never trust my compiler either), there is an option to display mixed C source and disassembly. Next you hit the appropriate key to cause a single step. The highlight moves to the 'next' thing to execute. If you're looking at assembly, the 'steps' are in instruction size. If you're just looking at the C source, the steps are one source line at a time. If you keep this up for a while you will see the source window adjust to keep your current position within the window.

Of course, you can scroll through the source window. Both programs give you a scroll bar. You don't even need your editor running to look at the source because you can look right here. *SDB* even has a 'find string' command to search for strings in the source file. Both programs number the source lines in the window, and the command languages for each allow specification of line numbers for commands like 'break'.

CodePRobe allows you to 'highlight' a line (by double clicking) which sets a permanent breakpoint on that line (although I could *not* find that information in the manual). Double clicking again removes the breakpoint. Unfortunately, at this point, the dialog window is not selected (the display window got selected when you clicked in it), and you cannot simply type **g** (for go) to get to the breakpoint. Ah well, minor annoyance for the ability to click on a line to select it for a breakpoint. *SDB* apparently does *not* use the mouse for any selection (other than scrolling).

Both programs have the ability to get out of the way when the program is running. On *CodePRobe*, the autoswap (as it is called) defaults ON instead of OFF like *SDB*. It can get annoying, but it is easy to turn off. Function keys will sort of exchange the debugger and the application windows in both programs.

Adapting my previous debugging techniques to these programs will undoubtedly take some time. For example, for several years now, I have used the breakpoint feature of debuggers almost exclusively at the entry to functions and on the locations to which they return. My first action when the break at a function entry is reached is to find and display the arguments. It turns out that both programs have a single command to do this 'ds' (*SDB*) and 'where' (*CodePRobe*), but these are really the stack backtrace commands. They list *all* of the functions which have been called thus far and all of their arguments also. I wish there was a simple command to show just the current situation - maybe the local storage, too.

Another command I would like is a 'break after this function returns'. I know how to find the return address, and I know how to put a breakpoint there, but a simple 'break once' there would be handy. Well, maybe I haven't *really* used these debuggers for long enough to become fluent in their use. Perhaps I will find a better way of debugging with these new tools.

Debugging technique

There is a temptation when using these programs to be less careful when writing code (and also when planning a debugging session). It has been my belief for some time that setting a breakpoint in a program is very much like taking a multiple choice test. That is, you should already have decided on what answer you think is 'right' before you look at the ones available. Well, with a computer and a breakpoint you only get one choice, but having decided *before* you push the button is the idea. Problems arise when you let your program tell you what the answer is, then you try to decide whether your program is right.

You've got a lot going against you at this point.

First, your baby (the program) just produced these results. How could anything so beautiful be wrong? Second, it has been repeatedly demonstrated that an authority figure producing a solution can convince even the most cynical of us that they have the correct answer. If you doubt that a computer is an "authority figure", try going over to your accounting department and explaining that the computer *must* have made a mistake on your latest paycheck. Too many times, I've seen computers intimidate people into believing that 'they' were correct when all common sense (an oxymoron if there ever was one) indicates otherwise. These computers and their programs even intimidate the authors of the programs... if we let them.

The most extreme example of this I've ever seen concerned a programmer who was writing a diagnostic for a brand new machine. He was dealing with two unknowns here. A machine which had *never* executed any instructions before, and a program which he had just finished. In his defence, I should point out that the engineers turned the machine over at about 10pm for the initial try at the diagnostic. It had been a long day, and sure enough, the diagnostic indicated that something was wrong with the machine. Not too surprising, since the machine had just executed its first instruction shortly before, so it should be obvious that this program had never been executed before.

Well, at any rate, as problems with the new design (hardware) were uncovered (as well as a few problems with the diagnostic itself), Carl began to take an approach that whenever a test would indicate a failure, he would immediately put the program counter back to the beginning of the test sequence and 'single step' though it. Unfortunately, he wasn't deciding beforehand what he expected. He was looking at results then saying yea or nay.

As you've probably guessed, there was indeed something wrong with the design of the machine and the combination of late hours and not 'picking answers' before results were available conspired to convince Carl that his diagnostic was wrong and the machine was indeed functioning correctly. This story has a happy ending: the machine - serial number 3 - was sent to a po-

tential customer who spotted this particular problem almost immediately. Seems that it is a common error for new computers to have, and the engineer/programmer at the customer site tested explicitly for it. I travelled to the customer's site about two days after this new machine got there. He was more amused than irritated, but pointed out that we'd better fix the computer.

I was almost incredulous that the error existed. I had (and still do) the greatest respect for Carl as a programmer (and a bug finder). I couldn't believe he hadn't tested for this boundary condition, so I called the home office and asked.

"Sure I checked that, why?" Carl responds. "Because the diagnostic passes here, but it fails the boundary test when we put it in by hand," I say. Carl again carefully examines the diagnostic. The test is indeed there, but the diagnostic is exactly backwards, the diagnostic is testing for the 'wrong' answer. Carl and I had a few laughs and a few beers over it when I got back to the home office, and I got the whole story, but it points out that careful testing is extremely important.

There are those who will disagree with me that you ought to carefully figure out what the 'answer' from a given program, function, source line (whatever) should be before you give the computer a chance to do it. In my opinion, that puts those of us who use that approach in the class of 'tinkerer' rather than programmer. I have no objections to tinkering with computers or running experiments with them. The thing I object to is running the experiment without any idea of what the results 'should' be.

There is another problem with 'single stepping' or even 'breaking on every iteration of the loop' without some idea of what the results should be: We are led into complacency by the monotonous correctness of the machine, and we let it seduce us into believing the wrong answer is right. (1 is prime, 3 is prime, 5 is prime, 7 is prime... <yawn> geez I guess all odd numbers are prime... 9 is prime, 11 is prime, 13 is prime... yup, all odd numbers *are* prime). Actually if the above were happening (and a physicist were looking at the results), he would just throw out 9 as experimental error. But programmers have to be a bit more cautious these days.

Computers are being used in more and more places. Sooner or later a computer software error is going to do some real damage (like kill someone). Let's *not* let it be our program. Testing software to eliminate bugs *is* possible, if it's carefully approached and implemented. Let's see if we can find the traps and avoid them. By the way, how many of you noticed that Richard Feynman's dissenting opinion on the Challenger disaster claims that the software production and testing was the *one* thing NASA was doing correctly?

Okay, next issue we will take a look at the more traditional debuggers available for the Amiga (unless, of course, Avant-Garde and Metadigm get their new stuff out), and we'll try to formulate some guidelines on how to improve *your* debugging.

Amiga Genealogy?

Now Shipping Your Family Tree

THE FIRST GENEALOGY DATABASE FOR THE AMIGA

- Standard Reports • Free Form Memo Area
- Supports Over 10 Million Family Members
- Easy To Learn and Use

YOUR FAMILY TREE

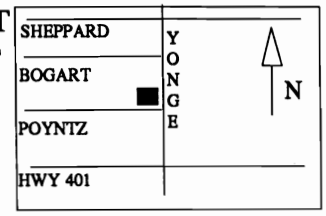
Contact Your Local AMIGA Dealer or Send $49.95 + 6% Sales Tax (PA. Residents Only). $4.00 Shipping and Handling (Within U.S.A.) to:
MICROMASTER, INC.
1289 Brodhead Road, Monaca, PA 15061
PH. (412) 775-3000

AMIGA is a Reg. Trademark of COMMODORE AMIGA

CANADA'S BEST SELECTION OF AMIGA - C64 - C128 SOFTWARE
COMSPEC HARD DRIVES AND BERNOULLI DRIVES AVAILABLE FOR THE AMIGA 500 AND 2000

Electronics 2001
COMPUTER PRODUCT CENTRE

4736 YONGE STREET
WILLOWDALE ,ONT
M2N 5M6
TEL: (416) 229-2700



AUTHORIZED COMMODORE DEALER SINCE 1979

D - Five Associates
19 Crosby Drive
Bedford, MA
01730-0523
(617) 275-8892

C-64/128 & peripherals

Tired of the high cost of computer repairs?
→ FLAT Labor charges
→ FREE Estimates
→ Warranty work

Commodore PC-10

Amiga 1000
* plus parts and sales tax

Also:
1764 to 512K: $61.95
128 64K vdc RAM: $40.00
NEW: C=1902 conversion to RGB-I: $40.00

Authorized Commodore Service Center

BRiWALL

Our Third Year In Business!

We Know How To Service Our Customers!

SOLID PRODUCTS & SOLID SUPPORT

P.O. Box 129 / 58 Noble Street
Kutztown, PA 19530

24 HOURS — TOLL FREE
1-800-638-5757

OUTSIDE USA CALL 1-215-683-5433

SATISFACTION GUARANTEED

Earn Bonus $$

No Surcharge On Charge Orders

Friendly Service

ACCESSORIES

Control Center	$ 79
Cover 1010 Disk Drive	8
Cover Amiga 1000 CPU	9
Cover Amiga 2000 System	25
Cover Amiga 1000 Keyboard	8
Cover Amiga 2000 CPU	9
Cover Amiga 2000 Keyboard	8
Cover Amiga 500	9
Cover Amiga Monitor	17
Flicker Master	14
Gender Changer/Parallel Adaptor	21
Lightpen for Amiga	89
Modem-Mini A1000	79
Modem-Mini A500/A2000	79
Modem Supra w/o cable	139
Modem Cable-RS232 to A1000	22
Modem Cable for A500	22
Okimate Plug N Print-Amiga	75
Ribbons	Call

BOOKS

1001 Things to do with Amiga	11
Advanced Amiga BASIC	15
Amiga BASIC Inside & Out†	20
Amiga DOS Express	20
Amiga DOS Inside & Out†	17
Amiga DOS Manual	20
Amiga DOS Reference Manual	19
Amiga Disk Drives Inside & Out	24
Amiga Hardware Manual	18
Amiga Machine Language†	17
Amiga Programmer's Guide	15
Amiga System Programmer's Guide	27
Amiga Tips & Tricks†	17
Amiga for Beginners	15
Beginner's Guide/Amiga	15
C Primer Plus	21
Elementary Amiga BASIC	13
Inside Amiga Graphics	15
Inside the Amiga With C	21
Intuition Manual	20
Kickstart Guide to Amiga	19
Machine Language-Compute	17
ROM Kernal : Exec	18
ROM Kernal : Lib/Dev	24
Abacus Disks each	12

GENERAL BUSINESS

Acquisition	189
Analyze 2.0	69
Becker Text	99
Critic's Choice	169
Data Retrieve	52
Data Retrieve Professional	199
Excellence	169
Family Tree (Your)	35
Flow	65
Goldspell 2	32
Haicalc	39
KFS Accountant	195
Keepin Track	45
Kindwords	69
Logistix	69
Magellian	139
MaxiPlan Plus	125
MaxiPlan 500	95
Microfiche Filer	69
Microfiche Filer Plus	125
Microlawyer	39
Money Mentor 2.0	69
Organize	49
P.H.A.S.A.R.	69
Page Setter	90
Professional Pages 1.1	249
Proscript for Prowrite	32
Prowrite v 2.0	75
Publisher Plus	64
Publishing Partner	$129
Reason	249
Scribble	55
Security Analyst	65
Shakespeare	149
Softwood File II SG	69
Softwood Write & File	69
Superbase Personal	59
Superbase Personal 2	105
Superbase Professional	235
TX ED Plus	59
Textpro	59
Viziwrite	59
Word Perfect	199
Word Perfect Library	99
Works (The)	125
Works - Platinum Edition*	175

EDUCATIONAL

Adventures of Sinbad	32
Aesop's Fables	32
All About America	39
Decimal Dungeon	32
Discovery Math	27
Discovery Spell	27
First Letters/Words	30
First Shapes	30
Fraction Action	32
Ghostly Grammer	32
Kid Talk	30
Kinderama	32
Linkword: French	24
Linkword: German	24
Linkword: Italian	24
Linkword: Spanish	24
Logic Master	32
Math Talk	30
Math Talk FRACTIONS	30
Math Wizard	32
Math-amation (College)	55
Mavis Beacon Typing	36
Perfect Score SAT	55
Planetarium*	49
Read A Rama	32
Speller Bee	29
Surgeon	34
Word Master	32

ENTERTAINMENT

4x4 Off Road Racing	29
AAArgh	25
Annals of Rome	27
Arkanoid	20
Auto Duel	35
Awesome Arcade Pak	36
Balance of Power	36
Barbarian	28
Bard's II: Destiny Knight	39
Bard's Tale I	33
Bard's II: Clue Book	12
Battle Chess	39
Better Dead then Alien	22
Beyond Zork	34
Black Cauldron	29
Blackjack Academy	27
Breach	27
Bridge 5.0	25
Bubble Ghost	27
Capone	$ 27
Carrier Command	32
Centerfold Squares	24
Chessmaster 2000	36
Clever & Smart	22
Corruption	32
Craps Academy	32
Defender of the Crown	33
Descartes	25
Double Dragon*	29
Dragon's Lair*	35
Dungeon Master	28
Earl Weaver Baseball	35
Earl Weaver's Commissioner's Disk	17
Ebonstar	27
Empire	36
F18 Interceptor	36
Faery Tale Clue Book	9
Faery Tales	32
Ferrari Formula One	36
Fire & Forget	29
Fire Power	19
Fire Zone	27
Flight Simulator 2	35
Fourth & Inches	29
GB Air Rally	25
Galactic Invasion	19
Galileo 2.0*	49
Gone Fish'n*	35
Guild of Thieves	29
Gunshoot	22
Hardball Baseball	29
Hunt for Red October	36
Hybris	27
Ice Hockey-Super Star	35
Ice Hockey: Wayne Gretsky	35
Jet	35
King of Chicago	33
Kings Quest III	36
Knight Orc	29
Leaderboard World Class	29
Leisure Suit Larry	29
Lord of the Rising Sun	33
Main Frame	28
Marble Madness	36
Master Ninja*	29
Mean 18 Golf	27
Mobieus	39
Obliterator	28
POW	29
Phasar Gun	42
Port of Calls	32
Portal	27
Pub Games*	27
Questron 2	36
Reach for the Stars*	29
Replay 4	29
Rocket Ranger	33
Roger Rabbit	33
Romantic Encounters - Dome	27
Rubicon Alliance*	24
SDI	33
Sargon 3	29
Scrabble	29
Sex Vixens Hint Sheets	5
Sex Vixens from Space	29
Shadow Gate	33
Shangai	27
Sherlock Holmes: Crown Jewels	29
Sky Fox 2	29
Space Quest 2	36
Space Station Oblivion	36
Spacecutter	$ 25
Star Fleet 1	39
Star Glider 2	32
Stellar Conflict	27
Strip Poker	29
Sword of Sodan	34
TV Sports Football*	33
Terror Pods	28
Test Drive	27
Thexder	25
Three Stooges	33
Tracers	27
Turbo	19
Twilight's Ransome*	27
Ultima 3	33
Ultima IV	39
Universal Military Simulator*	35
Virus	25
Wizard Wars	29
World Tour Golf	29
Zany Golf*	33
Zoom	20

CREATIVITY/GRAPHICS

Analytic Art	40
Animate 3D	99
Animation Effects	39
Animation Flipper	45
Animation Stand	39
Animator	85
Bumper Sticker Maker	37
Business Card Maker	37
Calligrapher v 1.05	79
Celebrity Cookbook	27
Comic Set-Funny Figures Clip Art	25
Comic Set-Science Fiction Clip Art	25
Comic Set-Super Heroes Clip Art	25
Comic Setter	69
Deluxe Paint 2	86
Deluxe Photo Lab	99
Deluxe Print 2	55
Deluxe Productions	139
Deluxe Video 1.2	86
Design Disks each	25
Digipaint 3.0	40
Digiview 3.0	129
Digiview Gold for A500/A2000	129
Director	49
Draw 2000	159
Express Paint V3	69
Fantavision	49
Flipside	25
Graphics Studio	39
Home Builder's CAD	135
Images	29
Intro CAD	52
Invision	99
Lights, Camera, Action	59
Modeler 3-D	69
Page Flipper	39
Page Flipper Plus FX	99
Page Render 3D	99
Page Sync	69
Photon Paint	65
Photon Paint Surface Disk	25
Photon Video Cel Animator	99
Photon Video Transport Controller	195
PrintMaster Fantasty Art Gallery	22
PrintMaster Art Gallery 1 or 2	22
PrintMaster Fonts & Borders	25
Printmaster Plus	$ 33
Pro Video Plus	199
Sculpt 3D	69
TV Show	65
TV Text	65
Turbo Silver Plus	139
Ultra CAD	169
Video Scape 3D 2.0	125
Video Titler 1.1	99
X-Specs	92
Zoetrope	89
Zuma Fonts-each	25

SOUND/MUSIC

Deluxe Music Construction	65
Dr T Midi Recording Studio	49
Dr. T KCS V1.6A	165
Dynamic Drums	52
Dynamic Studio	135
Instant Music	36
Midi Magic	99
Model A Midi Interface	49
Music-x	195
Perfect Sound A1000	65
Perfect Sound A500	65
Sonix	52
Sound Oasis	65
Sound Sampler	85
Soundscape Pro Midi Studio	129
Studio Magic	65
Synthia	65

UTILITIES/LANGUAGES

64 Emulator 2/Amiga 500	52
64 Emulator for Amiga 1000	52
A Talk Plus	52
AC BASIC v 1.3	135
AC FORTRAN	199
Amiga Workbench 1.3	24
Assempro	64
Aztec C Developer Compiler	195
Aztec C Professional Compiler	129
Aztec C Source Level Debugger	49
BBS PC/New Version	99
Benchmark Modula 2	129
Benchmark Modula 2-Simp A Lang	69
Benchmark Modula 2-IFF Library	69
Butcher 2.0	25
C.A.P.E. 64K	59
CLI Mate	27
Cygnus Ed Prof Text Editor	69
DOS-2-DOS	39
DSM	49
Diga	52
Disk-2-Disk	35
Diskmaster	39
FACC II	25
GOMF V3.0	27
GOMF V3 w/Hardware	49
Grabbit	20
Inova Tools	52
Lattice C + +	339
Lattice C MS-DOS/AmigaDOS Comp.	359
Lattice C V 5.0 Developer Pack	205
Mac Emulator*	call
On Line V2	42
Power Windows 2.5	59
Project D	35
Quarterback	45
True Basic Libraries each	35
True Basic 2.0	69
True Basic Runtime Pak	99
UltraDOS	39
Virus Infection Protection	33

IT IS OUR POLICY to stock what we advertise and carry the best products available at the best prices possible. OVER 500 PRODUCTS in stock now and more arriving daily!! IF YOU DON'T SEE IT LISTED, PLEASE GIVE US A CALL.

Products marked with * were not released at time ad was prepared. Please call for firm price and availability.

HARDWARE

CALL US!

Let us put a custom package deal together for you!

Please Call M-F, 9-4 EST

AmiGEN Genlock	$149
Bill's Boards Bus Expander	484
CMI VI-500 video adapter for A500	68
Color Splitter	75
Copy Stand	59

DRIVES

AIR 3.5" External	189
AIR 3.5" Internal	145
Mast 3.5" Internal	129
Mast 3.5" Twin External	259
Mast 3.5" Unidrive	149
Phoenix 22 Meg/A1000	579
Phoenix 22 Meg/A500	579
Phoenix 48 Meg/A1000	829
Phoenix 48 Meg/A500	829
Supra 22 Meg/A1000	659
Supra 22 Meg/A500	659
Supra 30 Meg/A1000	729
Supra 30 Meg/A500	729

ECE Midi 1000	49
ECE Midi 500/2000	49
Easyl 1000 Draw Tablet	369
Easyl 2000 Draw Tabler	369
Easyl 500 Draw Tablet	329
Flicker Fixer/A2000	489
Frame Grabber	549

GREAT VALLEY

A2000 20MB Hard Card SCSI	619
A2000 2MB Hard Disk Cntlr SCSI	309
A2000 30MB Hard Disk Card SCSI	679
A2000 40MB Hard Card Quantam	819
A2000 45MB hard Card SCSI	769
A2000 80MB Hard Card Quantam	1159
A2000-2/0 w/80MB Quantam Drv	1249
A500 20MB Hrd Drv/RAM Subsyst	699
A500 30MB Hrd Drv/RAM Subsyst	759
A500 40MB Q Hrd Drv/RAM Sbs	1020

Gen/One Genlock	719
IVS Trumpcard SCSI Controller	159
Live! A1000 Framegrabber	249
Live! A2000 Framegrabber	335
Live! A500 Framegrabber	285

MEMORY BOARDS

minimegs A1000 (OK)	165
Minimegs A500 (OK)	165
Pro RAM/A2000/OK	249
RC4 RAMcard/A1000/OK	199
RC4 RAMcard/A500/OK	199
Ronin M1000	519
Ronin M2000	519
Spirit-A1000 (OK)	240
Starboard II (OK)	269
Starboard II (1 Meg)	659
Starboard II (512K)	459

Panasonic 16mm Lens NO iris	22
Panasonic 16mm Lens with Iris	49
Panasonic W1410 Camera	189
Perfect Vision	169

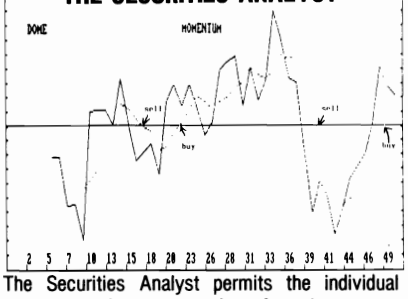
PRINTERS

Canon PJ1080A Clr Ink Jet	519
Hewlett Packard Desk Jet	779
Hewlett Packard Paint Jet	1150
NEC LC890 Poscript Laser	3499
Okimate 20	155
Okimate Plug & Print	55
Panasonic 1019i	239
Panasonic 1092	379
Star NX-1000R Rainbow	269
Toshiba Page Laser 12	3159
Xerox 4020 Color Ink Jet	1199

ProGEN Gen Lock	379
Ronin 68020-12 Microprocessor	189
Ronin 68030 adapter board	279
Ronin 68030 microprocessor	549
Ronin 68881-16 Math co-processor	189
Ronin 68881-20 Math co-processor	285
Ronin 68881-25 Math co-processor	579
Ronin H1000 Accelerator Board	345
Ronin Hurricane H2000	519
Subsystem 500	199
Supergen	689
Tiny Tiger Hard Drive Card	229

Actual Shipping Costs on Hardware

THE SECURITIES ANALYST



The Securities Analyst permits the individual investor to chart any number of stocks over an extended period of time. Using this package you can make intelligent buy-sell decision based on analysis of market trends. Analysis charts may be displayed on the screen or output to printer. A comprehensive manual on Technical Analysis of Stock Market Trends is also included. Great Buy!

INTRO OFFER $65

BONUS DOLLARS

EARN BONUS $$$ WHEN YOU ORDER!!! For every $50 of software and accessories that you order, you earn one bonus dollar. That's an additional 2% discount!! Use your bonus dollars on future purchases.

CATALOGS

We now have our ALL NEW AMIGA catalog, describing hundreds of products for your computer. Call or write for your copy of our AMIGA CATALOG today!!

BRiWALL

SOLID PRODUCTS & SOLID SUPPORT
P.O. Box 129 / 58 Noble Street
Kutztown, PA 19530
24 HOURS — TOLL FREE
1-800-638-5757

Circle 132 on Reader Service card.

As Captain Brad Stallion, you have been assigned a high priority mission by the Federated Government which has long been harried by a series of raids on the male population of its colonies. It is rumored that these raids are conducted by "The Tribe", a colony of beautiful female clones from the mysterious planet Mundo. Your mission is to locate the colony and destroy it! Please be advised that this is an ADULT adventure game!!

(Hint Sheets - $5)

ONLY $29

A Superior Word Processor!

Specifically designed for your AMIGA!

Experience

excellence!

No other product has all the powerful features required of a modern word processor, in a package sophisticated enough to use in desktop publishing! WYSIWYG, Spellcheck (as you type), Thesaurus, Multiple columns, Color, Graphics, Fonts, Math, Postscript Output and MORE!

SPECIAL PRICE! $139

(Good until 3/31/89)

SIZZLING HOT ENTERTAINMENT TITLES

Battle Chess	$39
Chrono Quest	35
Dragon's Lair	35
Dungeon Master	28
Heroes of the Lance (if released)	29
Pioneer Plaque	29
Pub Games	27
Rocket Ranger	33
Sword of Sudan	34
TV Sports Football	33
Ultima IV	39
Universal Military Simulator	35
Wayne Gretsky Hockey	35
Who Framed Roger Rabbit	33

BUY ANY 3 of the above titles and take an ADDITIONAL $5.00 off of your order!

(Offer good until 3/31/89)

ATTENTION! NEW AMIGA OWNERS

We want your business!

We will give you $10 off on your first software order if:

1.) Your order is $100 or more.

2.) You prepay with a check or money order.

3.)You send us a sales receipt for an A500, A1000 or A2000 dated between 12/1/88 and 1/31/89.

Offer good until 3/31/89.

Not valid with any other specials.

Come visit us! M-F, 9-5 EST, Sat. 9-12

New FAX (215) 683-8567

Prices subject to change without notice. Normally they don't, but this makes it legal. If in doubt, call!

ORDERING INSTRUCTIONS

For your convenience, you can place a phone order 24 HOURS/7 DAYS A WEEK on our TOLL-FREE LINE 1-800-638-5757. Price, Availability and Compatability Checks are also Welcome on our order line, Monday thru Friday, 9AM-4PM EST. AFTER HOURS, Orders Only Please!! When placing an order, please specify your COMPUTER MODEL, HOME & DAYTIME PHONE NUMBER, SHIPPING ADDRESS, METHOD OF PAYMENT and ITEMS ORDERED. To help us serve you better, please have all your information, including your CHARGE CARD number, ready before you call us.

ACCEPTED PAYMENT METHODS: We gladly accept payment by, PREPAID BY PERSONAL CHECK (will not hold for clearing) or MONEY ORDER: COD (continental USA only AND $4.00 ADDITIONAL); MASTERCARD; VISA; SCHOOL PURCHASE ORDERS. All payment must be in USA DOLLARS. THERE IS NO SURCHARGE FOR CHARGE CARDS and your card is NOT charged until we ship.

TECHNICAL SUPPORT

We do our very best to help you with your product selections, before you order and after you receive your product. General questions, price, and compatibility with your computer, etc. will be handled by our order staff at the time you place your order. BUT if you have specific, detailed questions about a product, printer compatibility questions etc., you will get the most help from our TECHNICAL SUPPORT LINE at 215-683-5699. Call Monday thru Friday, 9AM-4PM EST, and our trained tech staff will be happy to help you.

SHIPPING POLICY

ALL ORDERS received before 3PM EST will normally be shipped same or next business day. Out-of Stock items will be shipped by the same shipping method as original order, normally within 3 or 4 business days. All UPS shipments are sent SIGNATURE REQUIRED/NO DRIVER RELEASE. ADD the following shipping charges to your TOTAL software order.

UPS: GROUND-$4.00 (cont USA ONLY); AIR/RUSH-$7.00 (Includes Puerto Rico/Hawaii); OVERNIGHT-$12.00 PLUS $2.50 per item (must be received by 12 NOON)

PRIORITY MAIL: USA-$4.00 (includes APO/FPO); CANADA/MEXICO-$6.00; OTHER FOREIGN-$4.00 handling PLUS Actual Shipping (minimum $12.00)

RETURN POLICY [For software & accessories only]

We have a liberal return policy to better service your needs. Software piracy is a problem, but as long as our policy is not abused, we will continue to honor it. If within 15 days from the time you receive an item, you are not satisfied with it for any reason, you may return it to us for either a refund, exchange or open credit. REFUNDS ARE SUBJECT TO A10%RESTOCKING FEE PER ITEM RETURNED ($5.00 MINIMUM PER ITEM). A DEFECTIVE ITEM will be replaced with the same item (NO CHARGE... of course). EXCHANGES/OPEN CREDITS will gladly be issued for the FULL PURCHASE PRICE OF THE ITEM.

Proportional Gadgets With Embedded Numeric Feedback

42

by Hobie Orris

The proportional, or 'prop' gadget, sometimes referred to as a slider, has been part of mouse-driven user interfaces since the beginning of time. It consists of a (usually) long rectangular container, either vertically or horizontally oriented, within which a smaller rectangle (called the knob) moves freely up and down or back and forth when the user selects it with the mouse and drags it around. Its most common application has been to allow the scrolling of information inside a window, as in Intuition, GEM, Macintosh, and MS-Windows directory windows. More recently, application programs have adopted the prop gadget as an intuitive method of quantitative data input that provides easily understood visual feedback.

A well-known example of this last form of usage for the prop gadget is that of changing screen colours by adjusting the amounts of red, green and blue in the colour. In the case of the Amiga, as everyone knows, each of the 4096 colours is created by varying the intensity of each component hue by assigning it a value between zero and fifteen. Most paint programs provide the user with a means of changing a colour by sliding three prop gadgets corresponding to red, green, and blue, each with sixteen possible settings.

In the case of a colour palette the user may not be concerned with the actual values of the prop gadgets; he or she will be guided by the appearance of the colour rather than the numerical values that the slider positions represent. In other cases the value of a prop gadget, not merely its relative position, may be significant. One situation where a numerical value and a slider are both useful is where the user must enter a number within a finite range but the programmer doesn't want him or her to resort to entering it manually. In most current programs that require this sort of interaction, the commonest solution is to display the number below or next to the prop gadget and update its value as the user operates the gadget. Some programs leave the user to count out the value on a ruler-like scale drawn along the edge of gadget (like *Deluxe Paint*'s palette) or, even worse, to guess the value.

A notable exception to these (and the one which prompted this article) is an Atari ST program called *Cyberpaint*, which has been ported to the Amiga under the name of *Zoetrope*. Both of these programs display a large prop gadget along the bottom of the screen which selects frames to be displayed from an animation sequence. As the user slides the knob back and forth, the frames are displayed in forward or reverse order and (here comes the purpose of this article) the number of the frame in the sequence is displayed in the slider knob itself and is constantly updated as the position of the knob is changed. This placement has much to commend it: the user knows instantly the exact value of the prop gadget; the extra screen space used by other programs to display the slider value is saved; and the information in which the user is interested appears where the user's attention is directed - on the knob being moved. This is a simple and useful idea that any programmer interested in creating more readable prop gadgets should keep in mind.

Implementation of a simple numerical knob

The accompanying example program implements a single prop gadget that displays values between 0 and 99 inside the horizontally-sliding knob, changing as the user clicks on the gadget container or slides the knob back and forth. Rather than explain what the program does step by step, this article will merely outline the method used; you can follow the implementation details (which are fairly simple) in the code itself.

For those unfamiliar with the workings of prop gadgets, which are admittedly a little arcane, here is a brief orientation. To the programmer, a prop gadget is just like an ordinary single-click, or boolean, gadget except that it uses a few different options and it links to a 'PropInfo' structure. The PropInfo structure contains the information particular to prop gadgets, but only two fields of this structure are of any real use: for present purposes they are 'HorizBody' and 'HorizPot'. HorizBody is the increment size, or the amount the knob will move each time the user clicks on the gadget container, and HorizPot is the nu-

merical representation of the knob's position. Both of these are represented as 16-bit fixed-point fractions, which are awkward to use, but are a useful way of representing a large continuum of values without resorting to floating-point math.

In this representation scheme, the range of possible values is between 0 and 65535 (or 0xFFFF in hex), where 0 is the lowest value in the range and 65535 is the maximum value in the range. In the example program the chosen range of the prop gadget is 0 to 99, so a value in HorizPot of 0 corresponds to 0, 65535 corresponds to 99, and HorizBody is the increment size of 1/100th of 65535, or 655. A simple way of reading the prop gadget's setting would be to take the value in HorizPot and divide it by the value in HorizBody. The program's calculation of this value is done by a more accurate method, since using the above method, 65535 / 655 = 100, whereas it should be 99.

There are two features of the prop gadget called 'slider' in the program that must be defined in order for it to work like the one in *Zoetrope*. First, the AUTOKNOB option, which tells Intuition to define the size and shape of the slider knob, must be absent. Instead, the GADGHIMAGE option is set, an Image structure is created and the 'GadgetRender' field of the Gadget structure must point to it.

This Image structure contains the size, position and other attributes of the slider knob as well as a pointer to picture data which will be used to draw the knob image. The amount of storage required depends on the width, height and number of bit planes in the image, and is calculated in the routine init_slider(). Since the Text() routine will be used to draw to this area, this storage has to be allocated from CHIP memory.

The knob width of 20 pixels has been chosen to allow two digits of Topaz 8 text to fit inside, plus a border. If AUTOKNOB had been specified, then Intuition would have chosen a knob size proportional to the HorizBody size, 1/100th the size of the container, which would have been far too small to write into.

The second feature of the prop gadget is that the activation flags FOLLOWMOUSE and RELVERIFY are set. The first tells Intuition to send a MOUSEMOVE event whenever the knob is moved - this lets us update the display constantly, without having to stay in a busy loop and poll the knob position, making heavy demands on CPU time. The main loop calls set_slider() on every MOUSEMOVE event, which checks the current knob position and writes the appropriate number into the knob.

The last issue to be covered in this discussion is how to actually get the numerical value into the knob. In the routine init_slider(), a BitMap structure is initialized with the same dimensions as the previously-created Image structure, and the 'Planes' pointers are set to point to the Image data. The Image has only one pointer to data, so images with multiple bitplanes must have those planes next to each other in memory. The BitMap, on the other hand, has a separate pointer for each plane, so they can be scattered anywhere in (chip) memory. The init_slider() routine first determines the size of a single plane of the image and then figures out the offsets of the other planes. This example program uses two planes of image, but the code is general enough to support any number.

In order to write or draw to this new bitmap, it must have an attached RastPort structure, which contains the current font, pen colours, drawing position and other drawing attributes. Creating the BitMap and RastPort and linking them to the knob image in effect turns the knob into a little window that can be drawn into using the graphics library's routines. The only difference between drawing to the knob and any other window (other than its small size) is that the prop gadget must be refreshed using RefreshGList() or RefreshGadgets() before any drawing will be visible.

The numerical knob example is only a specific case of what is really a more general method. The same method, with minor modifications, could be used to display text strings in the slider knob which change according to the knob's position in the container, like a stereo receiver knob with settings for AM, FM, Tape and Phono. With a little more effort, the same method can even be used to display an animation as the user changes the knob position, by flipping the images in the image structure.

```
/*
 * slide.c -   An example program to demonstrate the use of a
 *         numerical proportional gadget knob.
 *         Compiled with Manx 3.6 and 32-bit integers.
 *
 *         Written Jan 6/89 by Hobie Orris.
 */

#include <stdio.h>
#include <intuition/intuition.h>
#include <graphics/gfx.h>
#include <exec/memory.h>

/* Offsets of the text drawing inside the slider knob */
#define TEXT_X 2  /* leave 2 pixels to the left as a border */
#define TEXT_Y 7  /* topaz 8 has baseline of 6. add 1 for a border */

/* these define the position of the gadget in the window */
#define SLIDE_X 20
#define SLIDE_Y 29

struct PropInfo sliderSInfo = {
   FREEHORIZ,  /* PROPINFO flags */
   0,-1, /* horizontal and vertical pot values */
   655,-1,  /* horizontal and vertical body values */
};

struct Image slider_image = {
   0,0,  /* XY origin relative to container TopLeft */
   20,9, /* Image width and height in pixels */
   2, /* number of bitplanes in Image */
   NULL, /* pointer to ImageData */
   0x0003,0x0000, /* PlanePick and PlaneOnOff */
   NULL  /* next Image structure */
};

struct Gadget slider =
{
   NULL, /* next gadget */
   SLIDE_X,SLIDE_Y,/* origin XY of hit box relative to window TopLeft */
   320,13,  /* hit box width and height */
   GADGHNONE|GADGHIMAGE,   /* gadget flags */
```

```
   RELVERIFY|FOLLOWMOUSE,   /* activation flags */
   PROPGADGET, /* gadget type flags */
   (APTR)&slider_image, /* gadget border or image to be rendered */
   NULL, /* alternate imagery for selection */
   NULL, /* first IntuiText structure */
   NULL, /* gadget mutual-exclude long word */
   (APTR)&sliderSInfo,  /* SpecialInfo structure */
   1, /* user-definable data */
   NULL  /* pointer to user-definable data */
};

struct NewWindow slider_window =
{
   140,60,  /* window XY origin relative to TopLeft of screen */
   360,60,  /* window width and height */
   0,1,  /* detail and block pens */
   GADGETUP|CLOSEWINDOW|MOUSEMOVE, /* IDCMP flags */
   WINDOWDRAG|SIMPLE_REFRESH|WINDOWCLOSE, /* other window flags */
   &slider, /* first gadget in gadget list */
   NULL, /* custom CHECKMARK imagery */
   (UBYTE *)"Slider example", /* window title */
   NULL, /* custom screen pointer */
   NULL, /* custom bitmap */
   NULL,NULL,  /* minimum width and height */
   NULL,NULL,  /* maximum width and height */
   WBENCHSCREEN   /* destination screen type */
};

struct TextFont *tf;
struct TextAttr default_font =
{
   (UBYTE *)"topaz.font", 8, 0, 0
};

struct RastPort srp;
struct BitMap sbm;
struct Window *win;
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;

struct Window *OpenWindow();
struct Message *GetMsg();
struct TextFont *OpenFont();
void *OpenLibrary(), *AllocMem();

main()
{
   struct IntuiMessage *imsg;
   ULONG class;
   int slide_value;

   /* Open libraries */
   openstuff();

   /* Set up the slider image before displaying it with OpenWindow */
   init_slider(&slider);
   if (!(win = OpenWindow(&slider_window)))
   {
      fprintf(stderr,"Can't open window\n");
      die();
   }
   set_slider(&slider);

   FOREVER
   {
      Wait(1L << win->UserPort->mp_SigBit);
      /* get only latest event to avoid queuing up mousemoves */
      while ((imsg = (struct IntuiMessage *)GetMsg(win->UserPort)) != NULL)
      {
         class = imsg->Class;
         ReplyMsg(imsg);
      }
      switch (class)
      {
         case CLOSEWINDOW:
            closestuff();
            exit(0);
            break;

         case MOUSEMOVE:
            set_slider(&slider);
            break;

         case GADGETUP:
            set_slider(&slider);
            slide_value = get_slider(&slider);
            printf("slider = %d\n",slide_value);
            break;
      }
   }
}


/* Read the value of the slider's horizontal pot and convert it to
 * a value in the intended range.
 */
get_slider(gad)
struct Gadget *gad;
{
   register ULONG longval;
   register struct PropInfo *sinfo = (struct PropInfo *)gad->SpecialInfo;
   register USHORT steps, val;

   steps = 0xffff / sinfo->HorizBody;  /* Figure out the range of values */
   longval = (ULONG)sinfo->HorizPot;   /* Get the current pot value */
   val = (USHORT)((longval * steps) >> 16);  /* Convert to range */
   return(val);
}

/* Write the converted pot value to the bitmap which is the knob image.
 * The imagery MUST be refreshed by RefreshGList, or else the number
 * will be out of sync with reality.  This also has the unpleasant
 * side-effect of causing the the gadget border to strobe.  To avoid this,
 * make the gadget PROPBORDERLESS and draw your own.
 */
set_slider(gad)
struct Gadget *gad;
{
   int val;
   char str[4];

   val = get_slider(gad);

   /* Convert the value of the prop gadget to a string and write it
    * to the rastport connected to the slider knob's rastport.
    */
   sprintf(str, "%2.2d", val);
   Move(&srp, TEXT_X, TEXT_Y);

   Text(&srp, str, 2);

   RefreshGList(gad, win, NULL, 1);
}


/* Initialize the structures that control drawing to the knob's image
 * data.  Also, open topaz 8 font, just in case the default font is
 * something larger (like topaz 9) which won't fit in the knob.
 */
init_slider(gad)
struct Gadget *gad;
{
   register struct Image *im = (struct Image *)gad->GadgetRender;
   register int i, plane_size;
   register long imdata;

   InitBitMap(&sbm, im->Depth, im->Width, im->Height);
   plane_size = RASSIZE(im->Width, im->Height);

   /* Allocate some image data memory.  Image data planes must occupy
```

```
     * contiguous memory, but BitMap planes aren't so particular.
     */
    if (!(im->ImageData = AllocMem(plane_size * im->Depth, MEMF_CHIP)))
    {
       fprintf(stderr,"Yikes! Can't allocate image data.\n");
       die();
    }
    /* This prevents the following addition from being calculated in
     * SHORTS rather than bytes.
     */
    imdata = (long)im->ImageData;

    /* Set the BitMap plane pointers to the knob image data. */
    for (i = 0; i < im->Depth; ++i)
    {
       sbm.Planes[i] = (PLANEPTR)(imdata + i * plane_size);
    }

    /* Initialize the RastPort structure */
    InitRastPort(&srp);
    srp.BitMap = &sbm;

    tf = OpenFont(&default_font);
    if (!tf)
    {
       fprintf(stderr,"Can't open font.\n");
       die();
    }
    SetFont(&srp, tf);
    SetAPen(&srp, 2); /* Draw in black */
    SetBPen(&srp, 3);
    SetDrMd(&srp, JAM2);
    SetRast(&srp, 3); /* Fill the knob in the background pen */
}

/* Just preliminary dull stuff */

openstuff()
{
    /* Note that we need at least 1.2 since RefreshGList is called */
    if (!(IntuitionBase = OpenLibrary("intuition.library",33)))
    {
       fprintf(stderr,"Can't open intuition library.\n");
       die();
    }
    if (!(GfxBase = OpenLibrary("graphics.library", 0)))
    {
       fprintf(stderr,"Can't open graphics library\n");
       die();
    }
}

closestuff()
{
    if (slider_image.ImageData)
       FreeMem(slider_image.ImageData, RASSIZE(slider_image.Width,
          slider_image.Height) * slider_image.Depth);

    if (win)
       CloseWindow(win);

    if (IntuitionBase)
       CloseLibrary(IntuitionBase);

    if (GfxBase)
       CloseLibrary(GfxBase);

    if (tf)
       CloseFont(tf);
}

die()
{
    closestuff();
    exit(-1);
}
```


AmiWare

Arts Education Entertainment

3640 Weston Rd., Weston, Ontario, Canada M9L1W2

	Cdn.$
ARexx	44.00
WShell	44.00
C.A.P.E. 68K	87.50
CygnusEd Professional	92.00
Lattice C v5.0 Developer	299.00
Lattice C++	450.00
Magellan	237.00
Manx Aztec C Developer	300.00
Modula 2 Developer	147.00
Multi-Forth	87.00
Power Windows 2.5	87.50
Flicker Fixer	799.33
US Robotics 9600	879.00
SCSI 40meg. 28 mse	689.00

If you are looking for a product that is not listed, give us a call.

Prices are subject to change without notice

Orders (416) 737-7385

24 Hr. BBS (416) 770-0124

-MAX™ The Build It Yourself Hard Disk System

- **No more painful waiting for icons to come up or large programs toload. -MAX- can provide over a 1/2 megabyte/ second reading rate with v1.3 FastFile System!**
- **Supports many 8 and 16 bit controllers, 68000 family CPUs and works with virtually all ST506/412 compatible drives.**
- **Compatible with most internal and external expansion devices.**
- **Once constructed, drive configuration is very quick and very easy using of a simple ASC11 text file and utilities provided.**
- **Typical cost for a 16 Megabyte hard disk system around $400.00. (Based on used Hard Disk of $100)**

Features: 1. Configurable for 1 or 2 controllers with 1 or 2 drives/controller. 2. Supports hundreds of megabytes/ drive and multiple partitions. 3. Can attach to and pass the 86 pin exp. bus for A500/A1000, or configure for A2000/ ZORRO II or ZORRO I 100 pin bus. 4. Supports OMTI, Adaptec, Western Digital and other controllers.

Includes: 1. Manual with illustrations, layouts and schematics, vendor list where parts can be purchased, (as no parts are supplied). 2. Device drivers, example configuration and Mountlist files. 3. Sophisticated hard disk exerciser/formatter/debugger. 4. Utility program with filecheck, diskcheck and archive bit functions. 5. Electronic Horizons' full and incremental backup utility.

To Order:
V3.0 available direct for $169.00. Send Cashiers Check or Money Order, COD (no Credit cards). PA residents add 6% sales tax. Outside USA: Money Order payable in US funds only, add $10.00 S & H. Business hours from 10 am to 9pm EST Mon-Sat. For tech, inquiries, please call between 6pm & 9pm EST Mon - Fri; 10 am & 5 pm Sat.

PALOMAX, Inc
424 Moreboro Road
Hatboro, PA 19040
215-672-6815

*-MAX- is a trademark of **PALOMAX**, Inc.*

Three Dimensional Fractals

Design and grow your own trees

by Cathryn and Eric Graham

Eric and Cathryn continue making improvements to Sculpt 3D and Sculpt Animate 4D. Recently they have been seen staring at the upper branches of the trees near their mountain home in south western Colorado. They can be reached at P.O. Box 8298, Durango, Colorado 81301. A disk containing the source code in this article, and a forest full of tree objects is available for $6.00, including postage.

Fractals have emerged from an obscure back alley of pure mathematics into the public's imagination. There cannot be an Amiga enthusiast who has not seen some example of the intricate beauty of fractal images. In this article we will show you how you can construct three dimensional fractal objects.

A fractal is a set of points, lines, surfaces, solids or higher dimensional objects that have such complexity that they break a dimensionality barrier and acquire a dimensionality that is no longer an integer. A curved line is usually considered to be a one dimensional object, because any point on the line can be described by a single number. A point on a two dimensional surface can be specified using two numbers. A fractal curve has intricate details on many size scales, so the apparent length of a portion of a fractal curve depends upon how closely you look. The outline of a sea coast behaves like a fractal. A bay may seem to be a mile wide when measured with a laser rangefinder, but when one walks along the shore and allows for the small scale irregularities of sandbars, the measurement could easily have doubled. A pedantic individual could measure around each pebble and come up with a larger value still, but even he did not measure around each individual grain of sand, or each atom on the surface of each sand grain.

Mathematicians have shown that they can treat idealized fractals in the same way that they have dealt with idealized lines or simple curves. It makes sense to describe a particular fractal as a one and a half dimensional object, for example.

Fractals are useful for describing many kinds of complicated behavior. To someone interested in computer graphics, they are useful for generating realistically complicated scenes. If you want an image of a mountain range, for example, you can either digitize a map of a particular range, or you can say that you do not need a specific mountain, just something that looks like a mountain. You still get to choose some of the properties of the mountain. Is it craggy, or just a rolling hill?

Solid models

In an earlier article in *TransAmi* (Volume 1, Issue 5), we showed how to make models of three dimensional shapes. Shapes are constructed from triangular faces. Each face is formed from three edges, and each edge is bounded by two vertices. The vertices lie at the corners of the triangular faces.

There are two stages to building a model. The geometrical stage consists of deciding where to place the vertices. The topological stage requires that you link pairs of vertices to form a triangular grid. We will follow the convention that whenever three edges are created in the form of a triangle, a face is implicitly created. A face also serves to hold information about an object, such as its surface colour or texture.

Finally, the description of the object must be converted into a data format that can be used by an object editor or a rendering program in order to produce an image. This was all described in the earlier article, which showed how to produce a file in the scene format of Sculpt 3D. You can use Sculpt 3D to render the resulting 3D image, or convert the file to another format with a utility like Syndesis' *Interchange* . You should refer to the previous article in order understand all the details of what follows. The examples used the C programming language and introduced linked list data structures for describing objects. An object structure was introduced that held pointers to a list of vertex, edge and face structures.

Vertices were created using a function:

```
struct vertex *newvertex(x,y,z,next)
double x,y,z;
struct vertex *next;
```

that specified the coordinates (x,y,z) of the vertex and served to link it to vertices that were created earlier. This function is all you need to use to supply the geometrical information.

The topological data is introduced with the following function:

```
link(object,v1,v2,properties)
struct object *object;
struct vertex *v1,*v2;
long int properties;
```

...which serves to generate an edge between two vertices. It also takes care of the possibility of a face being created if the new edge closes a triangle. In that case, the face is given surface properties as specified by the last argument.

A fractal curve

Let us see how easy it is to generate a fractal curve. The one we will make is known as the von Koch snowflake. The process is simple. Start with an equal sided triangle. Take each side and divide it into three equal pieces. Replace the centre piece with two sides of a smaller triangle. Keep repeating the process indefinitely, using each of the new sides that are formed each time. The end result is the snowflake fractal. In practice, we must always curtail the generation in the early stages, so we do not get a true fractal but only a complicated non-fractal. The snowflake can be generated by the following function:

```
snowflake(object,depth)
struct object *object;
int depth;
{int i;
 struct vertex *newvertex(),*v[3];
 double a,da,pi,sin(),cos();
 object->vertices=0; object->edges=0; object->faces=0;
 pi=3.1415927;
 da=2.0*pi/3.0;          /* 60 degrees in radians */

 for (a=i=0; i<3; ++i,a+=da)
    {v[i] = object->vertices = newvertex(0.0,
          cos(a), sin(a), object->vertices);
    }

 crinkle(object,v[0],v[1],depth);
 crinkle(object,v[1],v[2],depth);
 crinkle(object,v[2],v[0],depth);
}
```

The first executable statement of the function serves to initialize the object data structure. The loop provides the geometric information for three vertices arranged in the form of an equilateral triangle. Everything else is hidden in the function called crinkle. Here it is:

```
crinkle(object,v0,v4,depth)
struct object *object;
struct vertex *v0,*v4;
int depth;
{struct vertex *v1,*v2,*v3,*newvertex();
 double x0,x4,y0,y4,dx,dy;

 if (depth <= 0)         /* quick exit */
   {link(object,v0,v4,0L);
    return;
    }
```

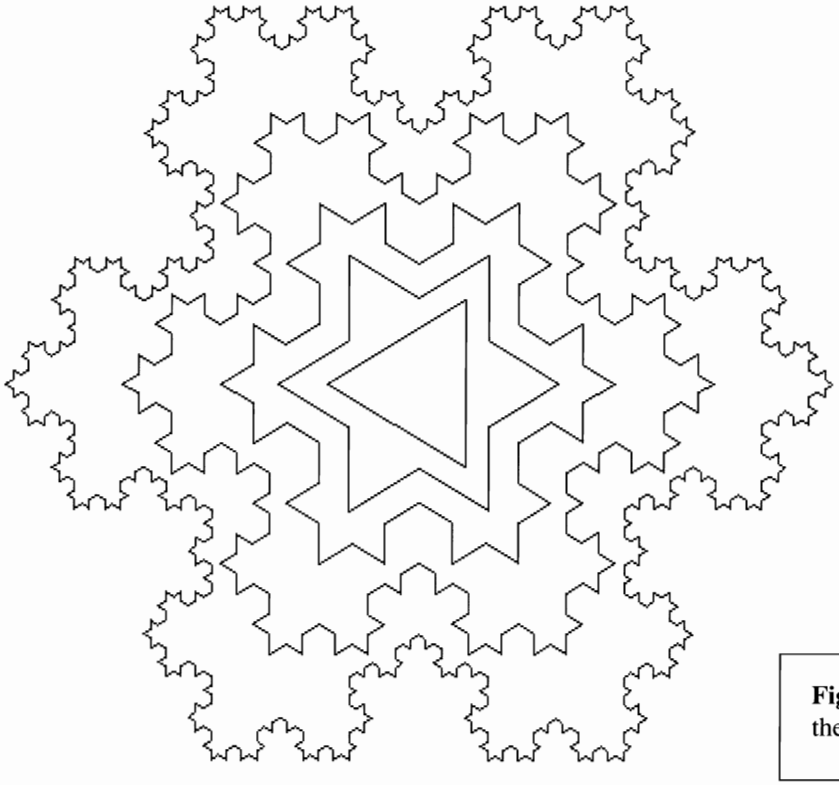


Figure 1: Successive generations of the von Koch snowflake

```
    x0=v0->pos[1]; y0=v0->pos[2];
    x4=v4->pos[1]; y4=v4->pos[2];
    dx=x4-x0; dy=y4-y0;

    v1=object->vertices=newvertex(0.0,x0+dx/3.0,
         y0+dy/3.0, object->vertices);
    v2=object->vertices=newvertex(0.0,
         x0+0.5*dx+0.2886751*dy,
         y0+0.5*dy-0.2886751*dx,
         object->vertices);
    v3=object->vertices=newvertex(0.0,x4-dx/3.0,
         y4-dy/3.0,object->vertices);

    crinkle(object,v0,v1,depth-1);
    crinkle(object,v1,v2,depth-1);
    crinkle(object,v2,v3,depth-1);
    crinkle(object,v3,v4,depth-1);
   }
```

The first part of the function provides for a quick exit by simply joining the two vertices with an edge. If the quick exit is not taken, two new vertices, v1 and v3, are generated that split the line between v0 and v4 into three equal parts. Another vertex, v2, is made so that it lies on an equilateral triangle with v1 and v3. Finally the crinkle function calls itself for each of the subintervals v0 to v1, v1 to v2, v2 to v3 and v3 to v4.

You have to take care when a function calls itself. The first danger is that an infinite regression may take place. This is avoided by keeping track of the depth of recursion, and providing a quick exit when the function is called too deeply. The second danger of recursion is that the object complexity grows exponentially with depth. The number of edges increases fourfold with each additional stage, so there is an upper limit to how deeply you can afford to go. The final danger is that the process of recursion itself uses memory to keep track of local variables and return addresses. This memory is taken from the stack area, which may be quite small. In our case, the second danger will almost always outweigh the third, because the exponential growth will eventually overtake the linear growth, unless the stack size is very small.

Recursion is an elegant technique for generating fractals, especially when the fractal is defined recursively. Figure 1 shows the first few stages of generating the snowflake. The image was obtained by rendering the scene file in wireframe mode. You can see how quickly the complexity of the snowflake grows with each additional stage.

The snowflake is confined to two dimensions, and we are more interested in solid three dimensional objects. Having generated a curve, you can use the facilities of your solid modelling program to convert the curve into a solid shape. Figure two shows what happens when you extrude the curve to produce a snowflake coin. You could also build snowflake pipes, donuts, helices or screws using the standard tools.

The basic method of recursively replacing a line by a more complicated set of lines can be performed in many ways. It is easy to change the code to generate other fractal curves, and then use them as a basis for making solid fractals.

A fractal surface

You may regard the way we turned a fractal curve into a surface as a bit of a cheat. Let us now build a genuine fractal surface, the next higher dimensional analogue of the snowflake.

The snowflake started with a triangle and kept on adding triangular bumps to each edge. Let us start with a tetrahedron, which is a pyramid with a triangular base, and add tetrahedral bumps to each face. If you do not already know the answer, you may care to speculate on the final shape of this fractal surface. Most people guess at some kind of spikey monster, but the surprise is that the fractal is bounded by a cube. This object has become known as the snowflake cube, and was the subject of an article by Scot Morris in the November 1988 issue of *Omni*. The illustrations for that article were generated on a Symbolics computer, but as we shall see, the Amiga is more than capable for the task. Here is a function that generates the snowflake cube:

```
snowflakecube(object,depth)
struct object *object;
int depth;
{struct vertex *v[4],*newvertex();
 double pos[4][3];
 static long int properties[] =
       {0xF0F0F000, 0x00FF0000,
        0xFF000000, 0x0000FF00,
        0xFFFF0000, 0xFF00FF00, 0x00FFFF00};
 int i;
 double a,da,pi,sin(),cos();

 object->vertices=0; object->edges=0; object->faces=0;
 pi=3.1415927;
 da=2.0*pi/3.0;          /* 60 degrees in radians */

 for (a=i=0; i<3; ++i,a+=da)     /* make the base */
    {pos[i][0]=0.0; pos[i][1]=cos(a);
     pos[i][2]=sin(a);}

 getapex(pos[3],pos[0],pos[1],pos[2]);

 maketetra(v,object,pos,properties[0]);

 addtetras(object,v[0],v[2],v[1],properties+1,depth);
 addtetras(object,v[0],v[1],v[3],properties+1,depth);
 addtetras(object,v[1],v[2],v[3],properties+1,depth);
 addtetras(object,v[2],v[0],v[3],properties+1,depth);
}
```

As expected, we construct the vertices for the initial tetrahedron and defer the real fractal generation to another function called 'addtetras', which adds tetrahedra to a triangle defined by three vertices.

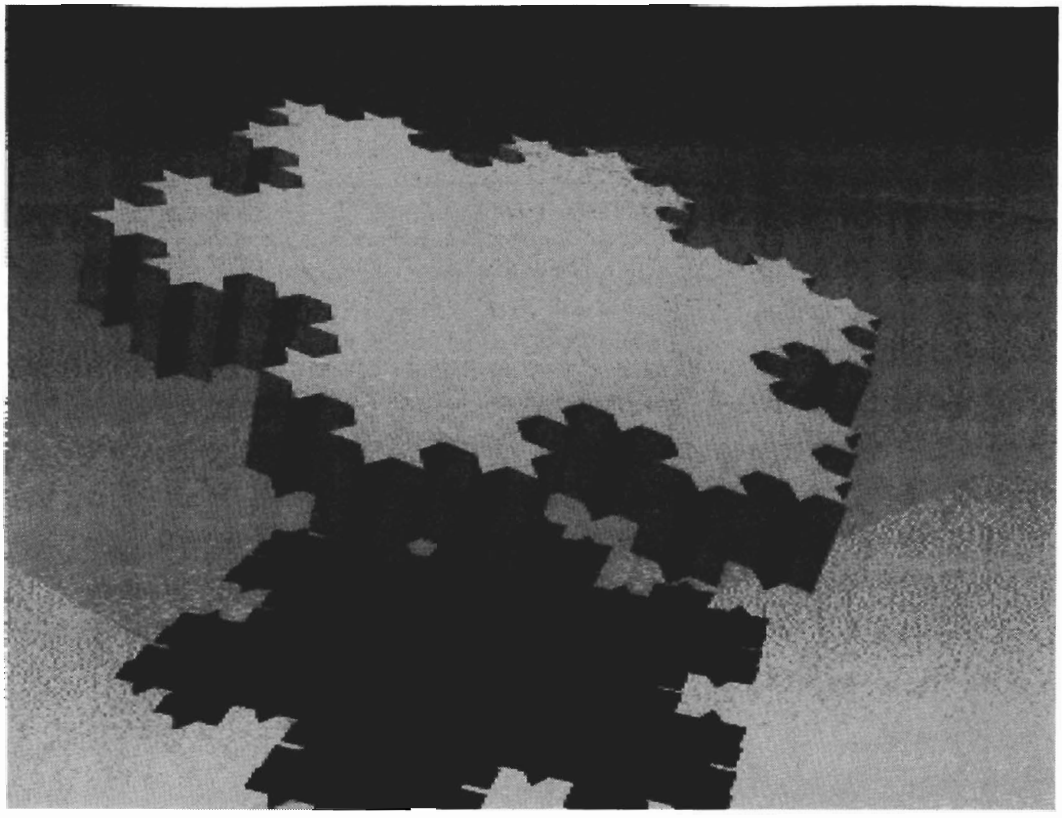


Figure 2: The von Koch snowflake fractal extruded into three dimensions and rendered with Sculpt-3D

The first tetrahedron is made by calculating the positions of the corners of an equilateral triangle. Then the function 'getapex' is used to calculate the position of the fourth corner. We will need this function again later. Finally the vertices are linked to form the surface of a tetrahedron by the function 'maketetra'.

When we built the snowflake curve, we left the actual edge construction until the deepest level of recursion, but this time we will build the shape by adding together solid tetrahedra. This actually makes the final object simpler and makes it easy to colour each generation of tetrahedra differently. The array called 'properties' contains colour data for seven generations, which would be enough for a very complicated object.

The function 'getapex' contains a little bit of solid geometry; here it is:

```
getapex(apex,base0,base1,base2)
double apex[3],base0[3],base1[3],base2[3];
{double a[3],b[3],c[3],perp[3],len1,len2,t,sqrt();
 int i,j,k;

 for (i=0; i<3; ++i)
    {a[i]=base1[i]-base0[i];
     b[i]=base2[i]-base0[i];}

 for (len1=len2=0.0,i=0; i<3; ++i)
        {c[i] = (base0[i]+base1[i]+base2[i])/3.0;
         j=(i+1)%3; k=(j+1)%3;
         perp[i]=a[j]*b[k]-b[j]*a[k];
         len2+=perp[i]*perp[i];
         t=base2[i]-base1[i]; len1+=t*t;
        }

 t=sqrt(len1/(1.5*len2));
 for (i=0; i<3; ++i) perp[i]*=t;

 for (i=0; i<3; ++i) apex[i]=c[i]+perp[i];
}
```

The position of the apex is first found by calculating the centroid of the three base vertices, then a direction perpendicular to the plane of the base is calculated as 'perp'. Finally, a distance along this direction is calculated that is appropriate for an equal sided tetrahedron.

So much for the geometry. The topology of a tetrahedron is much simpler. It is embodied in this function:

```
maketetra(v,object,pos,properties)
struct vertex *v[4];
struct object *object;
double pos[4][3];
long int properties;
{int i;

 for (i=0; i<4; ++i)
   v[i]=object->vertices=newvertex(pos[i][0],
       pos[i][1], pos[i][2], object->vertices);

 link(object,v[0],v[1],properties);
```

```
  link(object,v[1],v[2],properties);
  link(object,v[2],v[0],properties);
  link(object,v[0],v[3],properties);
  link(object,v[1],v[3],properties);
  link(object,v[2],v[3],properties);
 }
```

Finally we must provide the function that recursively adds tetrahedra; as you can see, it is quite straightforward:

```
addtetras(object,v0,v1,v2,properties,depth)
struct object *object;
struct vertex *v0,*v1,*v2;
long int properties[];
int depth;
{struct vertex *v[4],*newvertex();
 double pos[4][3];
 int i;

 if (depth <= 0) return;

 for (i=0; i<3; ++i)     /* midpoint of edges */
     {pos[0][i] = 0.5*(v1->pos[i]+v2->pos[i]);
      pos[1][i] = 0.5*(v2->pos[i]+v0->pos[i]);
      pos[2][i] = 0.5*(v0->pos[i]+v1->pos[i]);
     }

 getapex(pos[3],pos[0],pos[1],pos[2]);
 maketetra(v,object,pos,properties[0]);
 addtetras(object, v0, v[2], v[1], properties+1,depth-1);
 addtetras(object,v1,v[0],v[2],properties+1,depth-1);
 addtetras(object,v2,v[1],v[0],properties+1,depth-1);
 addtetras(object,v[2],v[0],v[3],properties+1,depth-1);
 addtetras(object,v[0],v[1],v[3],properties+1,depth-1);
 addtetras(object,v[1],v[2],v[3],properties+1,depth-1);
}
```

The image of a snowflake cube is shown in figure three, where its cubical shape is already becoming obvious. It would be a good candidate for an animation, if it were rotated. The general method of successively adding smaller versions of a simple shape can be used to generate many different fractal solids. If you design an interesting enough shape, it may even be named after you. I doubt that von Koch would have guessed that his name would almost become a household word.

Organic growth

As we mentioned earlier, fractals can be used for more than just generating interesting geometrical shapes, they can be used to simulate the world around us. A tree has some of the structure of a fractal: big branches split into smaller ones and so on until we reach tiny twigs. To misquote Joyce Kilmer:

> Programs are made by fools like me
> But only God can make a tree.

Undeterred, we will show you a program that will create something that is at least tree-like, and has some beauty of its own.

In the spirit of simplicity, let us grow a tree from a square

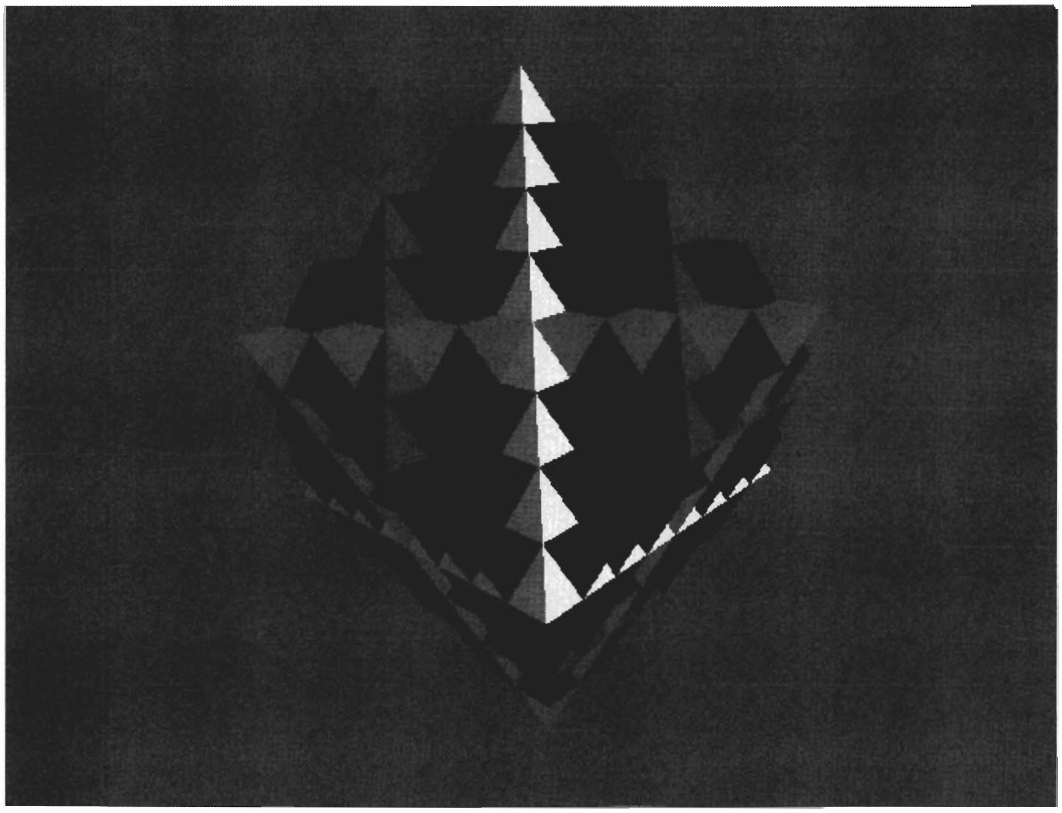


Figure 3:
A "true" three-dimensional von Koch snowflake, built upon the four faces of a tetrahedron.

cross section with the following rules:

1) A branch forks into sub-branches every so often.
2) Branches bend a little bit.
3) Branches twist as they grow.
4) Branches try to grow upwards.
5) Branches obey the law of gravity.
6) Branches get thinner towards the ends.

These rules are a trifle vague, but we will have no trouble converting them into C. The only rule that might seem superfluous is rule three about branches twisting. We only need that because of our square cross section; without it all our branches would wind up growing at definite angles. Here is our tree building program:

```
tree(object,treespec)
struct object *object;
struct treespec *treespec;
{struct vertex *v[4],*newvertex();
 int i;

 object->vertices=0; object->edges=0; object->faces=0;

 v[0]=object->vertices=newvertex(0.5,0.5,0.0,object->vertices);
 v[1]=object->vertices=newvertex(-0.5,0.5,0.0,object->vertices);
 v[2]=object->vertices=newvertex(-0.5,-0.5,0.0,object->vertices);
 v[3]=object->vertices=newvertex(0.5,-0.5,0.0,object->vertices);
 for (i=0; i<4; ++i)
        link(object, v[i], v[(i+1)%4], treespec->treecolor);
 grow(object, v, treespec);
}
```

This is really just a starter kit for tree building. The function just makes a square on the ground that serves as a growth ring for the tree, and then calls the grow function, that does the real work:

```
grow(object,v,treespec)
struct object *object;
struct vertex *v[4];
struct treespec *treespec;
{struct vertex *vt,*vring[4];
 double thickness, x[3], y[3], z[3], t[3],
      center[3], x1[3], z1[3];
 double pos[4][3];
 double veclen(),udran(),sqrt();
 int i;
 static double down[3]={0.0,0.0,-1.0};

 if (udran() < 0.5)        /* rotate by 90 degrees? */
   {vt=v[0]; v[0]=v[1]; v[1]=v[2]; v[2]=v[3]; v[3]=vt;}

 vecsub(x,v[1]->pos,v[0]->pos); thickness=veclen(x);
 if (thickness <= treespec->twigsize)
   {link(object,v[0],v[2],treespec->treecolor); /* close end */
    return;
   }
 vecsub(y,v[3]->pos,v[0]->pos);
 vecprod(z,x,y);
 vecnorm(x);
 vecnorm(y),vecnorm(z);  /* x,y,z are old growth axes */

 vsprod(t,treespec->droop*udran(),down);
 vecadd(z,t,z);      /* droop */
 vecnorm(z);
 axrot(x,y,x,y,udran()*treespec->twist);            /* twist */
 vecaddrandom(z,treespec->bend);                    /* bend */

 vecprod(y,z,x); vecprod(x,y,z);
 vecnorm(x);
 vecnorm(y),vecnorm(z);  /* x,y,z are new growth axes */

 center[0]=center[1]=center[2]=0.0;
 for (i=0; i<4; ++i) vecadd(center,v[i]->pos,center);
 vsprod(center,0.25,center);    /* middle of old growth ring */

 vsprod(t,treespec->growlength*thickness,z);     /* growth */
 vecadd(center,t,center);       /* middle of new growth ring */

 thickness*=treespec->taper;                    /* apply taper */

 /* to fork or not to fork */
 if (thickness > 2.0*treespec->twigsize)
  if (udran() < treespec->branchodds)
    {fork(object, v, center, x, y, z, thickness, treespec);
     return;}

 /* don't fork */
 for (i=0; i<4; ++i)
        {vsprod(t, ((i==1 || i==2) ? 0.5:(-0.5))*thickness, x);
         vecadd(pos[i], t, center);
         vsprod(t, ((i>1) ? 0.5:(-0.5))*thickness, y);
         vecadd(pos[i], pos[i], t);
        }

 for (i=0; i<4; ++i)
    vring[i]=object->vertices=newvertex(pos[i][0], pos[i][1],
        pos[i][2], object->vertices);

 for (i=0; i<4; ++i)
   {link(object, v[i], vring[i], treespec->treecolor);
    link(object, v[i], vring[(i+1)%4], treespec->treecolor);
    link(object, vring[i], vring[(i+1)%4], treespec->treecolor);
   }
 grow(object,vring,treespec);
}
```

Each invocation of the grow function starts with a growth ring and adds one stage of growth. It ends with either a new growth ring, or a pair of them if the branch is to fork. When a function calls itself as its last operation, that is called tail recursion, and it can be replaced by a loop. We have left it in a recursive form because we think that it is more attractive.

The first thing that the function does is to sometimes rotate the

input vertices by 90 degrees. By using the function udran(), which must return a random number between zero and one, we can do the rotation in a haphazard manner. The purpose of the rotation is to simplify the code when it comes time to fork. We only have to provide code for one of the two possible forking directions.

The function is provided with a structure called 'treespec' that contains all the statistical information needed to define an ensemble of trees. It is like the genetic code for a species.

```
struct treespec
        {long int treecolor;
         double droop;
         double twist;
         double bend;
         double taper;
         double growlength;
         double branchodds;
         double branchangle;
         double twigsize;
        };
```

We will explain what each part of treespec does as we continue to describe the grow function. The function makes extensive use of vector functions, like vecsub(), which subtracts two vectors. The thickness at the base of the branch is calculated by subtracting the vectors representing two adjacent corners of the square and then finding the length of the resulting vector.

If the thickness of the branch is less than 'treespec.twigsize', we say that the branch is too thin to be viable and close off the growth ring by adding an edge. This also truncates our recursion and enables the program to eventually stop.

Next we calculate three vectors, x, y, and z, such that each has a length of unity, and each is at right angles to the others. The growth ring lies in the plane of x and y, so the growth should occur roughly in the direction of z. Now to apply Rules 4 and 5, which say that trees like to grow up towards the light, but they must fight against gravity. The balance of these two effects is encoded in 'treespec.droop'. This should be a number whose size is less than one. If the number is positive, gravity will win and the tree will droop like a weeping willow. A negative value will enable phototropism to triumph over gravity, and the branches will tend to curve up, like a young fruit tree. The droop is applied by adding a vertically oriented vector to z and then renormalizing it. The size of the droop is randomized to make it look more natural.

Next some twist is applied to the branch by rotating the x and y vectors by a random angle whose maximum value is given by 'treespec.twist'. The angle is measured in radians so a value of 0.75 works fine. The actual rotation is performed by the following function

```
axrot(x1,y1,x,y,a)
double x1[3],y1[3],x[3],y[3],a;
/*      rotate a pair of orthonormal vectors (x,y)
        by an angle (a) to yield (x1,y1). */
{double t1[3], t2[3], t3[3], t4[3], sin(), cos();

 vsprod(t1,cos(a),x); vsprod(t2,sin(a),y);
 vsprod(t3,cos(a),y); vsprod(t4,-sin(a),x);
 vecadd(x1,t1,t2); vecadd(y1,t3,t4);
}
```

A certain amount of bending is introduced by adding a random perturbation to the z vector. The size of this is controlled by 'treespec.bend'. A value of 0.2 introduces a lifelike degree of waviness. After all our tampering with the z vector, it is renormalized and the x and y vectors are swung around so that all are again at right angles to one another. They now form the directions for a new growth ring.

The thickness of the ring is reduced by a tapering factor in 'treespec.taper'. Something a little less than one will work well, 0.95, say. The centre of the new growth ring is determined by advancing in the z-direction by an amount given by the product of the thickness of the branch and the parameter 'treespec.growlength'. Five is about right.

The decision as to whether the branch is to fork is made in such a way that the probability of a fork is given by 'treespec.branchodds'. A value of 0.5 will cause a fork every other growth stage, on average. A fork is only contemplated if the branch is at least twice the size of the smallest twig that is allowed. The actual fork operation is performed by the function called fork().

If no fork occurs, the last part of the grow function calculates a new growth ring and links it to the original ring in the form of a twisted tube of square cross section.

The fork function is as follows:

```
fork(object,v,center,x,y,z,thickness,treespec)
struct object *object;
struct vertex *v[4];
double center[3],x[3],y[3],z[3],thickness;
struct treespec *treespec;
{double x1[3],z1[3],t[3],pos[4][3];
 struct vertex *vring1[4],*vring2[4];
 int i;

 thickness*=sqrt(0.5);
 /* fork one way */
 axrot(x1,z1,x,z,-treespec->branchangle);
 vsprod(t,0.5*thickness,y); vecadd(pos[0],center,t);
 vecsub(pos[1],center,t);
 vsprod(t,thickness,x1);
 vecadd(pos[3],pos[0],t); vecadd(pos[2],pos[1],t);
 for (i=0; i<4; ++i)
        vring1[i]=object->vertices=
            newvertex(pos[i][0], pos[i][1], pos[i][2],
            object->vertices);
```

```
 /* fork the other way */
axrot(x1,z1,x,z,treespec->branchangle);
vring2[0]=vring1[1]; vring2[1]=vring1[0];
vsprod(t,-thickness,x1); vecadd(pos[2],pos[0],t);
vecadd(pos[3],pos[1],t);
for (i=2; i<4; ++i)
   vring2[i]=object->vertices=
      newvertex(pos[i][0],pos[i][1],
      pos[i][2],object->vertices);

link(object,v[0],vring2[3],treespec->treecolor);
link(object,v[3],vring2[2],treespec->treecolor);
link(object,v[1],vring1[2],treespec->treecolor);
link(object,v[2],vring1[3],treespec->treecolor);

link(object,v[0],vring1[2],treespec->treecolor);
link(object,v[1],vring1[3],treespec->treecolor);
link(object,v[2],vring2[2],treespec->treecolor);
link(object,v[3],vring2[3],treespec->treecolor);

link(object,vring2[3],vring1[2],treespec->treecolor);
link(object,vring1[2],vring1[3],treespec->treecolor);
link(object,vring1[3],vring2[2],treespec->treecolor);
link(object,vring2[2],vring2[3],treespec->treecolor);

link(object,vring1[0],vring1[1],treespec->treecolor);

link(object,vring1[1],vring1[2],treespec->treecolor);
link(object,vring1[0],vring1[3],treespec->treecolor);

link(object,vring2[1],vring2[2],treespec->treecolor);
link(object,vring2[3],vring2[0],treespec->treecolor);

grow(object,vring1,treespec);
grow(object,vring2,treespec);
}
```

If you look at this code closely, you can see that a square tube is constructed, but it is terminated by two square rings, shaped like a ridge roof on a house. Each roof panel serves as the starting point for more recursive growth. The thickness of a fork is reduced by a factor of the square root of a half, so that the total amount of wood is carried from the stem into the two branches. The angle by which each fork deviates from the main branch is given by 'treespec.branchangle'. Since it is given in radians, a value of 0.5 corresponds to about 30 degrees.

Finally we will show the vector functions that serve to simplify all the geometry:

```
vecaddrandom(v,size)
double v[3],size;
{double udran();
 int i;

 for (i=0; i<3; ++i) v[i]+=size*(2.0*udran()-1.0);
}

double veclen(a)
double a[3];
{double dot(),sqrt();

 return sqrt(dot(a,a));
}

double dot(a,b)
double a[3],b[3];
{double res;
```

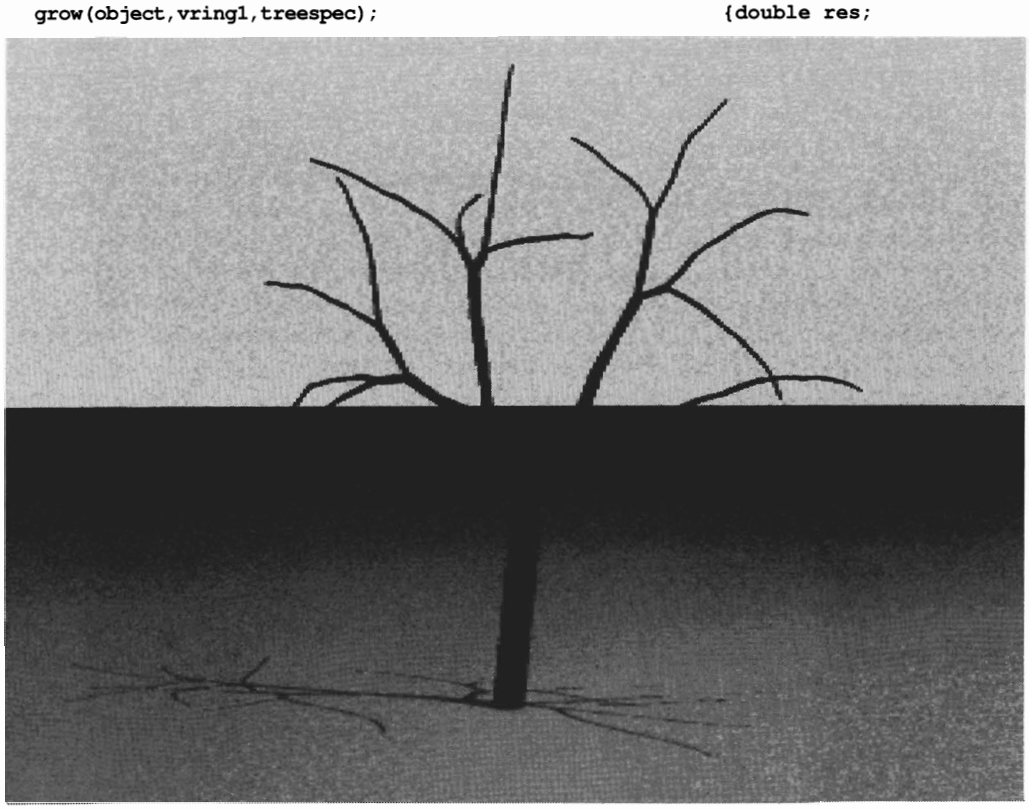


Figure 4: A sample tree fractal

```
 int i;

 for (res=0.0,i=0; i<3; ++i) res+=a[i]*b[i];
 return res;
}

vecsub(a,b,c)
double a[3],b[3],c[3];
/*      Vector subtraction a=b-c */
{int i;

 for (i=0; i<3; ++i) a[i]=b[i]-c[i];
}

vecadd(a,b,c)
double a[3],b[3],c[3];
/*      Vector addition a=b+c */
{int i;

 for (i=0; i<3; ++i) a[i]=b[i]+c[i];
}

vecprod(a,b,c)
double a[3],b[3],c[3];
/*     Vector product a=b^c  */
{double t0,t1;
 t0  = b[1]*c[2]-b[2]*c[1];
 t1  = b[2]*c[0]-b[0]*c[2];
 a[2]= b[0]*c[1]-b[1]*c[0];
 a[1]= t1; a[0]=t0;
}

vecnorm(a)
double a[3];
{double veclen();

 vsprod(a,1.0/veclen(a),a);
}

vsprod(a,b,c)
double a[3],b,c[3];
/* Product of a vector c and scalar b: a=b c */
{int i;

 for (i=0; i<3; ++i) a[i]=b*c[i];
}
```

Conclusion

Figure four shows one tree that was fashioned from the code. By varying the numbers in treespec, you can design different species of trees. Within each family, an infinite number of different trees can be built by changing the random numbers produced by the function udran(). Nor does our object building stop here; leaves could be added to the structure, and the rules of tree making changed in the never ending quest for more and more realism.

ARexx

The REXX Language for the Amiga

ARexx is a multitasking implementation of the REXX language, an elegant high-level language especially suited for macro-processing. Its clean, simple syntax makes REXX easy to learn for novices — and experienced programmers will appreciate advanced features like compound variables, INTERPRET instruction, and source-level debugging.

- Interactive, Interpreted Operation
- Exceptional String-handling Facilities
- Built-In Library with Over 75 Functions
- Built-In Source-Level Debugger
- Compact, Reentrant code — Only 32K

The Emerging Standard

ARexx is the *de facto* standard for inter-program communication. Software that supports the ARexx command interface can be customized, extended, and combined with other ARexx-compatible software in integrated applications. Watch for announcements of ARexx support from a growing list of Amiga software vendors!

Available Now ... Only $49.95

WShell

The Command Shell You've Always Wanted

WShell is the best command shell on the market, with features and convenience of use that go well beyond the CLI and 1.3 AmigaShell. And yet it's highly compatible with the CLI, so you don't have to learn a new command language to use it!

- Line Editing/Command History/Aliases
- Built-In Commands/Resident Commands
- Prompt String/Window Title "Variables"
- Concurrent Piping
- Transparent Support for ARexx Macros
- Script-bit Support for "Execute" Scripts
- Compact, Reentrant Code — Only 10K

ARexx Command Interface

With WShell you can use REXX-language macro programs like executables — even as filter programs in a piping system. Use the source-level debugging to single-step through a system of concurrently-running "piped" programs!

Available Now ... Only $50

Ask your Amiga dealer for ARexx and WShell, or order them directly from the author. Please include a check or money order plus $2/item shipping ($8 for overseas airmail) and any applicable taxes. Dealer inquiries welcome!

William S. Hawes
P.O. Box 308
Maynard, MA 01754
(508) 568-8695

Amiga is a trademark of Commodore-Amiga, Inc.

InterComputing Inc.

2100 N. Hwy 360, Suite 2101
Grand Prairie, TX 75050-1015

Orders only call : 1-800-622-9177 (now incl. Texas)

Customer Service and Order Status please use
1-214-988-3500

Call to recieve our new catalog!

Amiga is a trademark of Commodore-Amiga, Inc.

AMIGA

We are in business since 1984;
We support the AMIGA since 1985;
We will give you the best service possible at prices the other 'Discount-places' wont like - PERIOD !
Please call for a complete listing our products !
Member Better Business Bureau in Dallas

HARDWARE

Memory

8-Up Memorycard OK	145.75
8-Up Memorycard with 2MB	649.00
8-Up Pop-Simm 256k or 1MB	17.95
Starboard 0 MB A1000	239.75
Starboard 0.5MB A1000	399.75
Starboard 1 MB A1000	599.75
Starboard 0 MB A-500	239.75
Starboard 0.5MB A-500	399.75
Starboard Upper Deck Ok	44.75
A1000 256k Front Panels	199.00

Drives

FData-10 ext. Drive	139.75
int. 3.5" Drive A2000 only	145.75
HardFrame SCSI Card	239.75
Starboard SCSI Module	97.75
Quantum 80MB Drive	875.00
Supra 20MB HDisk	649.00
Supra 30MB HDisk	749.00
A501 - 512k w/clock	call

Don't Pay More !

Flexible Data Systems

FDATA-10 Disk Drive

This external 3.5" Drive is a very inexpensive unit to make the life with your Amiga easier.
The FData-10 is compatible with the original Amiga-drive but is quieter, smaller and has a longer cable (27inch) to connect it to the Amiga.

Oh yes, before we forget to tell you,
it's only $139.75 !

Video / Graphics

A-Pro Draw 12x12	499.75
A-Pro Draw 9x6	399.75
Amiga Live A 500	299.00
Amiga Live A 2000	329.00
Amiga Live A 1000	219.00
Digi View 3.0 NTSC	139.75
Digi View Adapter	19.75
Digi Droid	64.50
Easyl 500	299.75
Easyl 1000	349.00
Easyl 2000	349.00
Flicker Fixer	499.00
Flicker Master	12.50
Gen One Genlock	749.75
Imprint	399.75
Light Pen	99.75
Mitsubishi AUM 1371A	549.00
Panasonic 1410 Video Kit	259.75
Perfect Vision	175.00
Perfect Vision Color Splitter	85.75
Polaroid Palette	1995.00
Super Gen	699.75
X-Specs 3D	97.75

Great Hardware Buys

We now carry a wide variety of
MIDI-Equipment

Synthesizer	
Kawai K1 Synthesizer	749.50
Yamaha FB-01 Module	329.50
Percussions	
Alesis HR-16 Drum machine	379.50
Effects	
Alesis MIDIVerb II Digital Reverb	249.50
Mixer	
Kawai MX-8R Mixer (8 Channel / 2Effects I/O)	279.50
Midi	
ECE-Midi Interface	44.75
Gold Midi Internal for A2000	69.95
Midi Cable 8ft (your choice of 7 colors!!)	10.95

Sound / Midi

ECE-Midi w/ RS-232 thru	44.75
Future Sound Sampler	129.75
Soundscape Sampler	79.50
Soundscape Midi Interface	42.75
Perfect Sound A500/2000	59.75
Perfect Sound A1000	59.75
Midi Cable 5ft	8.75
Midi Gold for A2000 internal	69.95

Miscellaneous

Fuji 3.5" DS/DD Disks	17.75
Mouse Pad	6.95
Mouse Time for A1000	29.75
Multifunction Card	69.75
Phasor Gun	46.75
Processor Accelerator	149.50
Sony 3.5" DS/DD Disks	18.75
Supra Modem 2400 Baud	149.75

DESKTOP PUBLISHING

Comic Setter	66.75
Font Set I	24.75
Laserscript	29.75
Newsletter Fonts	29.75
PCM-Fonts Vol.1-3	27.50
Page Setter	89.75
Professional Page	239.75
Shakespeare	149.75
Studio Fonts	29.75
Zuma Fonts 1,2,3,4 each	24.75

MUSIC

Audio Master II	69.75
Deluxe Music	64.75
Dr. T's 4-OP Deluxe	99.75
Dr. T's DX-Heaven	99.75
Dr. T's Kawai K1 Libr.	99.75
Dr. T's KCS 1.6	166.75
Drum Studio	29.75
Dynamic Drums	47.75
Dynamic Studio	129.75
Laser Sound Vol. 1-3	27.50
Midi Recording Studio	44.95
Music Mouse	49.75
Sonix	47.75
Sound Oasis	66.75
Sound Quest Casio CZ	84.75
Sound Quest D-110	99.75
Sound Quest D-50	99.75
Sound Quest DX-7	99.75
Sound Quest DX-7 II	117.75
Sound Quest Generic	84.75
Sound Quest MT-32	99.75
Sound Quest Q-80-ESQ1	99.75
Sound Quest TX-81 C	84.75
Soundscape Pro Midi Studio	124.75
Soundscape Utilities I	39.95
Studio Magic	66.75
Synthia	66.75
Texture	99.75

Trade up!
We will trade in your Amiga 500, 1000 towards a new Amiga 2000 or 2500!
Please call for details.

Where are you, Video Toaster?

BUSINESS

Analyze 2.0	62.75
BEST Management System	299.75
Critics Choice	159.75
Cygnus ED	69.50
Excellence	189.75
Financial Plus	199.75
Flow	59.95
Gold Spell	29.75
Kindwords	66.75
Logistix	99.75
Maxi Plan 500	99.75
Maxi Plan Plus	129.75
Microfiche Filer	69.75
Microfiche Filer II	112.75
Money Mentor 2.0	59.75
Organize	66.75
Phasar 3.0	66.75
ProScript	32.95
Prowrite 2.0	72.50
Scribble 2.0	57.95
Softwood File II (GS)	79.75
Superbase Personal II	99.75
Superbase Professional	199.75
TexEd	22.75
TexEd Plus	54.75
Word Perfect Library	79.95
Wordperfect	197.75

GRAPHICS / VIDEO

3-Demon	69.75
Aegis Animator	82.50
Aegis Draw 2000	179.75
Analytic Art	39.75
Animate 3D	99.75
Animation Multiplane	56.75
Animator Apprentice	199.75
Butcher 2.0	21.75
Calligrapher new Version 1.05	87.75
Comic Setter	66.75
Deluxe Paint 2.0	94.75
Deluxe Photo Lab	99.75
Deluxe Productions	144.75
Deluxe Video 1.2	94.75
Director	44.95
Dynamic Cad	299.00
Fantavision	39.95
Flipside	39.75
Forms in Flight II	77.75
Gallery 3D	43.75
Interchange	29.75
Intro CAD 2.0	49.75
Kara Fonts	52.75
Lights, Camera, Action	52.75
Modeler 3D	72.50
Pageflipper Plus F/X	99.75
Photon Paint	66.75
Pixmate	49.75
Prism Plus	44.75
Pro Board	379.75
Pro Net	379.75
Pro Video Plus	199.75
Sculpt 3D	69.75
Sculpt 3D (PAL)	69.75
Sculpt 4D	399.75
Silver	99.75
Terrain Disk for Turbo Silver	19.75
TV Show NTSC/PAL	62.95
TV-Text	62.95
Turbo Print	32.75
Turbo Silver	129.75
Video Effects 3D	129.75
Video Titler	99.75
Videoscape 3D	119.75
Window Print II	22.75
X-CAD	359.00
Zoetrope	99.75

GAMES

4th and Inches	29.75
Andromeda Mission	27.75
Amegas	19.75
Arazok's Tomb	24.75
Arkanoid	19.75
Balance of Power	33.75
Barbarian	24.75
Battle Chess	31.75
Black Shadow	25.75
Capone	29.75
Carrier Command	29.95
Centerfold Squares	22.50
Championship Basketball	29.75
Championship Baseball	27.75
Championship Football	29.75
Championship Golf	27.75
Corruption	27.95
Cube Master	19.75
Defender of the Crown	35.75
Destroyer	27.75
Ebonstar	29.75
Emerald Mine	17.75
European Scenery Disk	19.75
Faery Tale Adventure	29.75
Ferrari	32.75
Final Assault	31.75
Fire and Forget	28.75
Flightsimulator	39.75
Foot Man	22.75
Foundation Waste	27.75
Gee Bee Air Rally	27.75
Global Commander	29.95
Grand Slam (Tennis)	29.75
Guild of Thieves	29.75
Hollywood Poker	24.75
Hybris	27.75
Interceptor	34.75
Into the Eagles Nest	24.75
Jet	32.75
Jewels of Darkness	9.75
Karate Kid II	27.75
Knight Orc	29.75
Major Motion	27.75
Menace	21.95
Mindwalker	27.50
Obliterator	29.75
Offshore Warrior	28.75
Pioneer Plague	27.95
Plutos	19.75
Portal	34.75
Ports of Call	29.75
POW	25.75
Q-Ball	19.75
Realm of the Warlock	24.75
Return to Atlantis	34.75
Road Raider	27.75
Roger Rabbit	29.95
Scenery Disk 7/11	19.75
Scenery Disk Europe	19.75
Scenery Disk Japan	19.75
Shadowgate	29.75
Shanghai	27.75
Silent Service	27.75
Silicon Dreams	9.97
Sindbad	33.75
Skychase	27.75
Space Cutter	19.95
Space Quest	32.75
Starglider	29.75
Starglider II	29.75
Starwars	27.75
Street Cat	17.75
Street Sports	14.95
Strip Poker	27.75
Strip Poker Data #4 or #5	14.95
Surgeon	29.75
Terror Pods	29.75
Test Drive	29.75
The 3 Stooges	34.75
The Pawn	29.75
Torch 2081	19.75
Uninvited	33.75
Vampires Empires	29.75
Virus	19.75
Vyper	22.75
Wintergames	14.95
World Class Leaderboard	29.95
World Games	27.75
Zoom	21.75

UTILITIES

Anti-Virus IV	call
Arexx	32.75
CLI Mate	27.75
Disk Mechanic	56.75
DOS-2-DOS	42.75
Demonstrator	24.75
Encore	44.95
EZ-Backup	32.75
Facc II	19.75
Fine-Print	32.75
Marauder II	27.75
Project D	32.75
Quarterback	44.95
Raw Copy	39.95

BOOKS

AMIGA Graphic/Sound	17.75
Amiga Dos Manual	19.50
Desktop Video	14.95
Hardware Reference Man.	19.75
Intuition Reference Man.	19.75
ROM-KERNEL: Lib.&Dev.	27.75
ROM-KERNEL: Exec.	19.75

MISCELLANEOUS

A-Talk III	67.95
Intellitype	34.75
Online	44.75
The Works	129.75

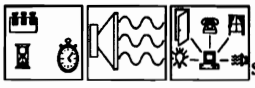
New Software in Stock

4x4 Off Road Racing	Micro Fiche Filer II
Battleship	Orbiter ?
California Games	Photon Paint II ?
Dragon's Lair	ProScript
Dr. T's Kawai K1 Lib.	Pro Sound Designer
Dungeon Master	Sculpt 4-D
Falcon	TV-Sports Football
Lattice C 5.0	Zuma Fonts 4

PROGRAMMING - LANGUAGES

A/C Basic Compiler	129.75
A/C Fortran (AB-Soft)	179.75
Aztec C Developer	199.75
Aztec C Professional	145.75
Aztec Source Level Debug.	59.75
Benchmark Amiga Library	64.75
Benchmark C-Library	64.75
Benchmark IFF Library	64.75
Benchmark Modula-2	134.75
DSM MC68000 Disassembler	42.75
Lattice-C ++	395.00
Lint	69.00
MCC-Pascal	79.95
Modula-II Commercial	175.00
Modula-II Developer	89.75
Modula-II Standard	59.75
Power Windows 2.5	56.75
True Basic	69.75

Hardware of the month.



Intruder Alert!
Home or business alarm.
External BSR/X-10 control.
Synthesized computer speech.
Sampled alarm effects!

Completely multitasking with hooks to start other programs. Easy to use graphic operating system. There is potentially unlimited use in this outstanding new product! **$115.00**

And as allways;
we've got the most
'customer friendly' Terms :
S/H $3.95
30$ Minimum Order
NO Credit Card Fee !
but
in TEXAS add 7% TAX !
All prices are subject to change without notice !

$3.95 S/H to Cont. USA only!
APO address, please call.
RMA# required on all returns.

Sliding Window Protocols

Almost everything you wanted to know about link protocols but were afraid to ask

by Matthew Dillon

Matt Dillon is an Electrical Engineering senior at Berkeley. He has been working on software and hardware hacks beginning with his first microcomputer - a Commodore PET. He is widely known in the Amiga community for his DOS shell and his text editor, DME.

Actually, a general discussion on the subject of network and link protocols would fill several books. Unfortunately, most authors on the subject have a tendency to concentrate on summarizing the umpteen already-existing network protocols without saying much about the actual theory required to write the software to drive one of them. The well known ISO layering model for networks is often summarized and used to compare various types of networks (as it was meant to do), but that is as deep as most authors go.

In this article I will take the reverse approach and discuss the actual implementation of a link protocol I wrote for the Amiga called 'DNet'. Meaning for this article to be practical, the only theory I will discuss will be that which pertains to DNet. In accordance with the title of this article, I will spend much more time on the implementation of the low level packet protocol than on other parts of DNet.

There will be some confusion here over my naming of DNet. Properly, it should have been named DLink, and if I had had more wisdom in the distinction at the time I began writing it, I would have named it that. I didn't, though, and the name stuck. Fortunately, DNet is not one of your everyday file transfer protocols. For you theoretical types, DNet is a (reliable) symmetrical link protocol able to handle multiple connections in both directions using a BSD-style client-server model. DNet uses a sliding window protocol with four slots at the lowest level with packets ranging from six to around 200 bytes in length. Being a link protocol and not a network, DNet is well suited for communication between any two machines it has been ported to. You can run it between two Amigas, two BSD4.2/4.3-compatible UNIX machines, or between an Amiga and a UNIX machine.

The layering of DNet

1) The physical layer, i.e. serial port communications. DNet simply uses the *serial.device* here.

2) A low level sliding window packet protocol guarantees a reliable data stream in both directions. The RESTART protocol is embedded here (i.e. link start-up).

3) A high level stream protocol encapsulates control commands and data associated with individual channels. The control commands are used to open, close, and send data over connections.

4) The library interface between client and server programs and DNet. In the Amiga implementation, this is a link library that provides a simple command interface (open, close, read, write, etc...) to programmers who wish to write clients and servers for DNet.

In my discussion of these various layers below, I will refer mainly to the Amiga implementation. The BSD4.3 UNIX implementation is similar, but having two Amigas at home (an A1000 with lots of extra junk on it and an unadorned A500), I wrote the Amiga side first and tested the network between the two machines. I then copied the source to my Berkeley account and modified it to work with UNIX.

The low level packet protocol

The easiest place to start is with the lowest layer, which is roughly equivalent to any of the well known file transfer protocols. This layer, which I will call the LLPP, is responsible for getting the data given to it to the other machine without error. This layer communicates with the *serial.device* to transmit and receive raw data. The carrier detect line is used to determine when a connection is lost, and the baud rate is used to determine when the ACK is expected for a packet (i.e. various timeout conditions).

The LLPP implements what is known as a sliding window protocol. Most people are familiar with simple packet protocols where machine A sends a packet to machine B and then waits for an ACK from machine B before sending the next packet. If the ACK does not come back within a reasonable amount of time machine A assumes the packet was garbled and resends it. Each packet contains a sequence number that allows the receiving machine to identify duplication of data and which data that packet's control commands pertain to. There are two major problems with this approach. (1) There is a delay between machine B's reception of the last byte of machine A's packet, and machine A's reception of the ACK before it can begin sending the next packet. Thus, the serial line is not being utilized for a short period of time and throughput is reduced. (2) If A's packet gets munged or B's ACK gets munged, a large delay ensues during which the serial line is idle that could have otherwise been used to send further data; throughput is again reduced, this time to a major degree.

Before I delve into the art of writing a sliding window protocol, I would like to note a very minor modification that works extremely well with a sliding window protocol and sometimes even improves non-windowed protocols. The concept is known as a CHECK packet. Instead of the transmitter resending a packet for which it did not get any response, it sends a small CHECK packet (the same size as an ACK or NAK) asking the receiver to respond with an ACK if it already has the packet or a NAK if it does not have the packet. This prevents the transmitter from resending an entire data packet when the receiver has already got it. There is a real life situation in which this comes up constantly: Some networks are so slow that DNet and other protocols will time out even though the receiver gets the packet and the ACK will eventually get back to the transmitter. In this case, if the first method had been used (just resending the packet), one would see the transmitter sending every packet at least twice, making for extremely inefficient usage of the bandwidth.

A sliding window protocol is the logical next step in finding a solution to the various problems with older protocols. A sliding window protocol simply means that machine A does not wait for machine B to ACK a packet before it begins sending the next packet. I had better define some terms before going into a discussion:

Window Size - The difference between the packet ID of the next packet ready to be transmitted and the lowest packet ID yet to be ACKed. The Window Size constantly changes and is limited to some maximum.

By the way, if the packet ID ranges from 0 to 7 (as DNet's does), and the lowest unACKed packet is #5, and the next ready packet is #1, we have 'turned over'. The window contains 5, 6, 7, and 0, and thus its size would be 4.

Window Size Limit - A window is not allowed to grow beyond some maximum size, even if there is only one unACKed packet.

UnACKed Packets - The number of unacknowledged packets is independent of the window size, but never larger than the window size.

I will explain the sliding window protocol from machine A to machine B, with A transmitting packets and B acknowledging them. Keep in mind that DNet actually runs two sliding window protocols in a symmetrical fashion, one in each direction.

As I said above, A will begin sending the second packet before it gets the ACK for the first. A will continue to send packets until its Window Size reaches its Window Size Limit. Now A *has* been processing what ACKs or NAKs have come in during all of this. The processing of an ACK can make the current Window Size smaller if it is ACKing the lowest unACKed window.

```
packet # (== window ID):  0 1 2 3 4 5 6 7 0 1... (rolls over)
N = not ACKed                N A N A
A = ACKed                    ^       ^
                           window  window (window size = 4)
                            start    end
```

Let the maximum window size be 4, so at this point the transmitter can no longer transmit new packets. The next packet that would have been transmitted is #6. Two of the packets have not been ACKed: #2 and #4. The transmitter will retry packet 2 by either asking the receiver if it got it and retransmitting the packet only if a NAK comes back, or by simply retransmitting the packet in question. The transmitter will probably also retry packet #4 while waiting for the ACK/NAK for #2. This continues until #2 is ACKed. The moment #2 is ACKed, the window start pointer can move past positions 2 and 3 (since it was ACKed) and would rest at position 4 until packet #4 is ACKed. So as soon as #2 is ACKed the window size is reduced to 2 and the transmitter can begin sending packets again (#6 and #7). The idea in a sliding window protocol is that when a glitch occurs, the protocol should resolve the glitch before its transmitter reaches the Window Size Limit and must pause. As long as the protocol does not reach this limit it can transmit continuously.

The Window Size Limit depends on several items depending on how you implement the sliding window protocol. In terms of memory usage, the transmit side is limited by the maximum number of unACKed packets it can allow (since it must keep buffers for those in case it needs to resend them). The receive side, however, is limited by the maximum window size, since it must keep around packets until it can sequence them. That is, if packet #2 is not ACKed and packets 3, 4, and 5 have been ACKed, the receive side cannot dump the data in 3, 4, and 5 until it gets valid data for 2 or it will be dumping data to the next level out of order. Finally, the Window Size Limit cannot be larger than half the maximum range of window IDs available. DNet uses three bits to store its window ID, and with only eight possible values the Window Size Limit is four. The Window Size Limit for the entire protocol is, of course, the minimum of these three parameters. Small machines with no virtual memory or machines running over slow links (e.g.

2400 baud modems) must usually set the Window Size Limit to something reasonable as DNet does. Larger machines on faster links or on networks can afford to run the protocol with much larger limits, allowing for a constant and efficient stream of data no matter how many line errors occur.

Why can't the Window Size Limit be larger than half the maximum number of window IDs? Simply put, the transmitter's idea of the current window is not necessarily the same as the receiver's idea. For instance, in the example above let us say the receiver had actually gotten packet #2 and its ACK back to the transmitter got garbled so the transmitter thought the receiver did not get packet #2. The receiver's idea of the sliding window would be (4,5) and the transmitters would be (2,3,4,5). The transmitter then tries to DNet CHECK for packet #2. The proper response by the receiver is, of course, an ACK. But look, the receiver's window is (4,5). How does the receiver know that the CHECK 2 is for a *previous* packet that it has obviously already ACKed or for a *later* packet that it has not even seen? The *only* reason the receiver knows that the CHECK 2 is for a previous packet is due to the extra slop in the number of window IDs available. The receiver can calculate the worst case limits of the transmitter's idea of the window:

```
packet # (== window ID):        0 1 2 3 4 5 6 7 0 1 ...
            RECEIVER                  N A
            TRANSMITTER             N A N A
Receiver's calculation of transmitter's
window (worst case min/max):      ? ? ? ? ? ? ?
                                 older pkts  newer pkts
```

Since the receiver has not ACKed packet #4, it knows the transmitter's window cannot be any less than (Limit - 1) to the left of ID 4 or any greater than (Limit - 1) to the right of ID 4. Any CHECK's that occur (Limit - 1) to the left of the receiver's idea of the window start (4) are obviously already ACK'd and any CHECKs that occur (Limit - 1) to the right of (4) that are not within the receiver's window (4,5) have obviously not been received yet and should be NAK'd.

Any CHECKs that occur within the receiver's window are either ACK'd or NAK'd depending on the current state of that window ID. That is, both the transmitter and receiver keep track only of the ACK'd/Not ACK'd state within their window. Since we have allowed 2*Limit of slop there is no overlap and the receiver can determine that the transmitter's CHECK 2 packet is referring to something the receiver has already ACKd rather than something the receiver has yet to ACK.

If the Limit had been just *one* larger, the following scenario could occur (limit is now incorrectly five with eight IDs):

```
packet # (== window ID):        0 1 2 3 4 5 6 7 0 1
            RECEIVER                  N A
            TRANSMITTER           N N N A N A
Receiver's calculation of transmitter's
window (worst case min/max):    ? ? ? ? ? ? ? ? ?
                               older pkts    newer pkts
```

If the transmitter were to send a CHECK 0 packet, the receiver would not know whether the transmitter were referring to an older packet that the receiver had already ACK'd, or a newer packet that got garbled. Maximum confusion; protocol death.

With a sliding window protocol, machine B can determine almost immediately if it has missed a packet, long before A would otherwise determine that B did not get the packet. Simply put, if B expects to get packet #N and gets packet #N+1 instead, it can immediately send a NAK for packet #N. The non-windowed protocols can only send NAKs if they time out or 'see' a garbled packet (it depends on the protocol). The so-called NAK packet, which is almost useless in a non-windowed protocol, becomes an active method of error recovery with the sliding window protocol. B can truly know that it missed a packet and send the NAK before A has exhausted its window (has sent all the packets it can before pausing). Thus A's transmission train is not broken and throughput depends only on the packet overhead (header, checksum, etc.) and number of packet errors, with no penalty beyond that. This solves the second problem mentioned for non-windowed protocols.

DNet is known as a *symmetrical* protocol. That is, there is no master or slave in the protocol. Both sides run exactly the same algorithm. Thus, both sides transmit a data stream and receive the other side's data stream.

A DNet low-level packet looks like this:

SYNC CTL CTLCHK [LENH LENL DATA CHK1 CHK2]

SYNC 0xFF (1 byte)

This is a sync byte used to find the starting point of packets and to help prevent misinterpretation of data as the starting point for a packet.

CTL sssdrccc (1 byte) bit order 76543210

sss sequence number 0-7. You need twice as many sequence numbers as packet windows to handle CHECK packets that might occur for packets that have already been processed and forgotten.

d 1 = DATA part exists for this packet, else 0 = is a 3 byte packet.

r reserved for future use, must be 0.

ccc Control code 0-7
- 0 Reserved
- 1 DATA packet
- 2 CHECK packet (3 bytes)
- 3 ACK packet (3 bytes)
- 4 NAK packet (3 bytes)
- 5 RESTART packet (used for protocol start-up)
- 6 RESTART-ACKNOWLEDGE
- 7 Reserved

CTLCHK = (SYNC(0xFF) << 1) ^ CTL (i.e. FE ^ CTL)

A check byte for the control byte.

The following is sent only if the 'd' (data exists) flag is set in the control byte:

LENH	MSB of # bytes of data
LENL	LSB of # bytes of data
DATA	N bytes of data
CHKH	MSB of data checksum
CHKL	LSB of data checksum

Here is the checksum algorithm (I did not make this up, but saw it somewhere... it is supposed to be pretty good):

```
chkbuf(buf, bytes)
register ubyte *buf;
register uword bytes;
{
   register uword i;
   register ubyte c1,c2;

   for (i = c1 = c2 = 0; i < bytes; ++i) {
      c1 += buf[i];
      c2 += c1;
   }
   c1 = -(c1 + c2);
   return((c1<<8)|c2);
}
```

DATA packet

A DATA packet contains anywhere from one to around 200 bytes of data. The number 200 was picked as a maximum because UNIX machines generally cannot buffer more than 256 bytes of data before getting overruns. The extra 56 bytes of leeway provides enough time for the UNIX process to wake up and read the data. The UNIX DNet protocol process attempts to sleep for as long as possible, calculating when it expects the entire packet to have been received before waking up and reading it. This is necessary to maintain a high scheduling priority as real time response is required for the smooth running of the protocol. This maximum is one of the problems with DNet. Properly, my RESET protocol should pass the maximum packet size its particular implementation can stand and the RESET-ACKNOWLEDGE should return a similar number by the other machine. The minimum would then be used. Amigas do not have the limitation of 200 byte packets and people running them at 19.2KB have not gotten the high efficiency one gets at lower baud rates. Even so, DNet gets much higher throughput at high baud rates than any other protocol I know.

Redundant DATA packets received are ACK'd and then thrown away. As soon as a valid sequence of data packets is received (e.g. if you miss 1 and get 2 and 3, a valid sequence 1, 2, 3 will occur when you finally get 1 back), their contents are sent, in order, to the next level up.

When sending DATA packets, the expected time before the ACK is received is calculated and a couple of seconds are added. A timeout sequence is begun and, if not aborted (by a valid ACK or NAK coming in for that packet), a timeout will occur causing the sending of a CHECK packet to figure out what happened. Note that in most cases an ACK or a NAK will be returned before the timeout occurs.

CHECK packet

A CHECK packet is sent by one machine when the other machine has failed to respond with either an ACK or a NAK to a DATA packet sent to it. The packet is sent every couple of seconds until the other machines responds with an ACK or NAK. An ACK means that the other machine has indeed received the packet in question and the first machine need not resend it; a NAK means that the other machine has not received the packet in question and the first machine should resend it.

ACK packet

Whenever a machine receives a valid packet, even a redundant one (it had already received that packet before), or receives a CHECK packet and had already received the valid packet specified by the sequence number, it returns an ACK packet.

NAK packet

Whenever a machine determines it missed a packet, or receives a CHECK packet and has not already received the valid packet specified by the sequence number, it sends a NAK packet. Upon reception of a NAK packet, a machine resends the packet in question.

RESTART packet

A RESTART packet is sent when DNet is initially started up. It causes the receiving machine to close all connections to external clients and servers and to reset its protocol. The receiving machine returns a RESTART-ACKNOWLEDGE packet. DNet being a symmetrical protocol, either side may initiate a RESTART.

When DNet is initially started up on both sides, both will initiate a RESTART sequence simultaneously, which is okay.

In all but the RESTART packet, the packet's sequence number (0-7) is used to determine which command effects what.

General sequencing of data

Transmissions from machine A to machine B contain DATA and CHECK packets sourced by A, and ACK/NAK packets replied to B from A due to B's sending DATA and CHECK packets to A.

Because this is a symmetrical protocol, the B-to-A direction is exactly analogous. It contains DATA and CHECK packets sourced by B, and ACK/NAK packets replied to A from B due to A's sending of DATA and CHECK packets to B.

The high level stream protocol

So far, we have discussed no more than the reliable transmission and reception of a single data stream in each direction. This is as far as the file transfer protocols we know and hate so well go, and DNet goes quite a distance beyond it. DNet allows you to open several distinct two-way streams simultaneously over a single serial line, and it does so by multiplexing those multiple streams along with other control information via the High Level Stream Protocol. This high level protocol is superimposed on the low-level packet protocol's DATA packets. At this point all data are assumed to be reliable, so no checksums are needed for the high level data structures being sent and received.

DNet uses what is known as a client-server model for creating connections between processes on the two linked machines. Servers 'listen' on ports for connections. A server is a program with a purpose - for example, a terminal program that waits for the remote machine to try to open a talk window. Each server in the system is given a known port number. A client is the program that initiates a connection to the server on the remote machine and then talks to that server over the connection. For example, I have written the following servers for DNet:

```
8192 dnet:amiga/bin/scopy      ;File Transfer server
8195 dnet:amiga/bin/sterm      ;Terminal Window server
8196 dnet:amiga/bin/scli       ;Shell Server (needs pipe:)
8197 dnet:amiga/bin/sloadav    ;Load Average Server
8198 dnet:amiga/bin/sprint     ;Printer Server
                                 (direct copy) (new)
```

If I were to dial up another Amiga or UNIX host and we both ran DNet, I could say this from a CLI:

```
1> putfiles some_file
```

Putfiles is a client program that attempts to connect to port 8192 on the destination machine. Port 8192 is the *scopy* server, which accepts the connection. The programs then talk to each other with the *putfiles* client sending the specified file or files to the *scopy* server. What we have just done is transfer a file from one machine to another through the generalized client-server model. DNet is intelligent in the way it handles servers. If a remote machine attempts to connect to a port for which no server is running, DNet will attempt to look up the proper server in a file, start it going, and then attempt to connect to it.

But since DNet allows multiple connections simultaneously, we could also open a terminal window by running the *fterm* client, which connects to the *sterm* client on the remote machine, and the two users (let's say you call up your friend and both run DNet over a modem) could talk to each other while the file transfer is going on. Not only that, but you could also start a file transfer in the other direction... the modem's Rx and Tx LED's would be solid on! And through all of that you could talk to the guy on the other end through one or more terminal windows. This is the advantage of using DNet over anything else currently out there... you can do lots of things simultaneously. There are other servers available, such as a simple *sprint* server I wrote (this is a simple printer server - something new that the first release of DNet did not include), and the *scli* server, which allows one to bring up a remote Amiga CLI as a connection. The *scli* server is very minimal though, and a big hack, but only because I had problems getting a CLI to talk to DNet rather than a console as it usually does.

The big question is how to manage all of these connections. Connections are opened (a client connects to a server), closed, and data is transferred in both directions. Multiplexing all of these control and data operations is the job of the high level stream protocol. High level stream protocol packets require no checksums and no acknowledgement (unless you need to pass some state back) because the low level protocol has already guaranteed us a reliable data stream in both directions.

A high level data packet consists of a special control byte and then further information:

mxlllccc (1 byte, bits 76543210)

m = 0:
command is SCMD_DATA, length is xlllccc (i.e. 0-127), data follows

m = 1, x = 0:
command is ccc (0-7), length is lll (0-7) (see below)

m = 1, x = 1:
length is (lllccc << 8) | next_byte, command is SCMD_DATA, data follows

This format is not as general as it could be, but exists for a purpose. If I had wanted a general format I would have had one command byte, two data length bytes, and *N* bytes of data (raw data or control structure data). Encoding a single character of data would have taken four bytes instead of two.

The reduction of overhead is essential only for small amounts of data, and was done to increase response time for single characters on an idle line. Notice that only SCMD_DATA (data passed from a client to a server or vice versa) can have lengths larger than seven bytes. Other commands will contain some data, such as a channel number of result code, but the total structure size will never be more than seven bytes.

The ccc commands are as follows (0-7):

0 **SCMD_SWITCH**: Switch which channel the data stream is being sent to. SCMD_DATA stream packets do not contain a channel number, for efficiency reasons, so whenever data from a channel other than the 'current' channel must be sent, we must first switch to that channel with the SCMD_SWITCH command. Again, this is to reduce the overhead for typing single characters on an idle line.

This command is the heart of the data multiplexing capability that allows DNet to handle many distinct connections simultaneously.

1 **SCMD_OPEN**: Open a new channel. A client on one machine wishes to connect to a server on the other. The DNet protocol driver on the client end assigns a free channel number and then sends this high level packet to the other end. The channel number is used to identify this distinct stream/channel, from other active channels in the system. The channel's state will be set to 'PENDING' until an SCMD_ACKCMD stream packet comes back. It is possible for both machines to open a new channel simultaneously and attempt to assign the same channel number. This will be detected and will cause both ends to retry opening the connections with a different channel number.

2 **SCMD_CLOSE**: This command closes a channel. For example, when you hit the close gadget on a DNet terminal window, the connection is closed. The DNet protocol itself is still running, of course; you have simply closed one of the *N* client-server connections.

3 **SCMD_ACKCMD**: This command is returned by machine B (A) in response to an open or close command issued by A (B). In the case of an open, the data part holds a return code informing A (B) if the open succeeded or failed. In the case of a close, it informs A (B) that it will no longer attempt to write to the channel.

Additional information, such as receive and transmit priority, is also passed.

4 **SCMD_EOFCMD**: This command allows one side of the connection to send an EOF to the other side without actually closing the connection. It informs the other end that this end is through transmitting data, but still allows the other end to send data back before closing the connection.

5 **SCMD_QUIT**: This command is used to quit out of DNet - kill the protocol driver itself. Rightly, it should be a low level protocol command but I ran out of command IDs.

6 **SCMD_IOCTL**: This command is used to send I/O control commands to the other side of the connection. For example, when you resize a terminal window an IOCTL will be sent to the other side, and if that side is a UNIX host capable of changing its TTY parameters, they will be modified to handle the new window size. IOCTLs are synchronous with the data stream, and the effect of IOCTLs is totally dependent on the clients and servers.

(8) **SCMD_DATA**: This is the actual data sent from one machine to another for a specific connection. The data written by a client is placed in an SCMD_DATA packet and sent to the server, and vice versa. There is no real command ID for it because it is a special case (re-read the format of the protocol given above).

Library interface

Since DNet uses message ports on the Amiga and not normal file handles, a set of library routines exists that the client programs can call to work with DNet connections. The major calls are given below. The actual distribution includes several other calls to handle non-blocking reads, sending EOFs, receiving IOCTLs, etc. With just a few commands, all the power of DNet is placed at the programmer's whim.

listenport = DListen(portnumber): When a server is run, it makes this call to inform DNet that it is listening on a given port number for new connections. A server is run by the DNet driver itself. Once run, it stays resident (usually idle) until the DNet driver itself is killed. On UNIX systems, the servers stick around until the machine reboots or the user kills them. But this is not a problem, they take very little space and virtually no system resources (just a UNIX domain socket).

New connection requests arrive on the message port. These requests are *not* interpreted by the server code itself, but on detection the server should make either a DListen() or DUnListen() call to process them.

(void) DUnListen(listenport): When a server is about to exit, it makes this call to inform DNet that it is going away and no new connections can be accepted on the given port number. Any not-yet-accepted connections pending on the listen port are returned with an error.

chan = DAccept(listenport): When a server sees a message on the listen port, it may call this routine to accept the connection. (This causes DNet to send the SCMD_ACKCMD packet mentioned above.)

(void) DNAAccept(listenport): If a server does not want to accept a pending connection, it may call this routine. (This causes DNet to send the SCMD_ACKCMD packet mentioned above, but with an error code)

chan = DOpen(host, portnumber, transmitpriority, receivepriority): A client wishing to open a new connection to some remote server makes this call. The host is currently not implemented and should be NULL. The remote server is identified by its port number, and a transmit and receive priority is given.

The transmit priority determines where DWrite requests will be queued going into the DNet server, and the receive priority determines where DWrite requests will be queued going into the remote DNet server.

actual = DRead(chan, buf, bytes), and **actual = DWrite(chan, buf, bytes)**: These two routines are used to read and write data to an open channel. Both client and server may make these calls.

Non-blocking calls also exist, plus a way of getting signalled when new data has arrived, allowing for completely asynchronous operation.

(void) DIoctl(chan, cmd, val, aux): This routine is used to send an IO control message to the other end. Both the client and server may make this call.

(void) DClose(chan): This closes a connection. <Poof>, gone.

A brief extension into networks

Just to show how close to a real network DNet is, I will briefly describe what it would take to run DNet as a network rather than a link protocol. The primary modification would be the addition of two fields to the low level packet interface: a FROM and a TO field, perhaps a byte each, indicating the source and destination machine for the packet. Running DNets would each have their own address and only accept packets with their address as the TO field. There would be a DNet running on machine X for each machine it has active connections to. That is, the window protocol would be on a **machine A ↔ machine B** basis and not on a network-wide basis. Various protocol timeouts would also have to be added (those have to be added anyway) in case a machine goes down in the middle of a transfer. If we ignore routing between several networks, this would be sufficient to implement a client-server model on a local net.

Security with DNet

How secure is your system if you are running DNet? It all depends on which servers you allow to run. I originally wrote DNet to allow me to connect to my account on a Berkeley UNIX machine and run multiple shells, file transfers, and dump things directly to my Epson from my UNIX account. Thus, I was the only one on both sides. To use DNet between two friends over a modem would also be fine. But to use DNet between two strangers over a modem is not so fine - in particular, using DNet to run a BBS.

Still, you can do it. The BBS operator would have to modify the code for the servers to limit what the potentially hostile user might try to do - start up fifty terminal windows, for instance, or download into the BBS's C: directory, or hang the connection (DNet currently does a packet-retry forever or until CD is lost). The current implementation of DNet's file transfer program can download or upload entire directory trees to anywhere on the destination machine. That would have to be changed and limited. These are all things I plan to do in the future - if enough people are interested. I have found DNet to be an invaluable tool both for talking to my UNIX machine and fixing a friend's problems via a remote CLI. All in all, a good return on my investment of time writing it.

* * * * *

*Editor's note: DNet is freely distributable, and will be available on a **TransAmi** disk in the near future. Matt is currently making some enhancements and fixes, particularly in the area of security discussed just above; he expects an updated version to be ready in about a month. Meanwhile, the current distribution of DNet will be included on the disk for this issue if space permits.*

AMIGA RGB CABLES
CUSTOM RGB
AMIGA TO 4 BNC
SONY KV1311 23F-34F
SONY MULTISYNC 23F-9M
NEC MULTISYNC 23F-9M
MAGNAVOX ANALOG 23F-6MDIN
MITSUBISHI MULTISYNC 23F-25M
ATARI ANALOG ADAPTER 23F-13F
ALL 29.00 TO 39.00
Shipped Blue USA

Typical Non-Standard Cable
Call For Quotes On Weird or Unusual Cables
REDMOND CABLE
For The Dealer Nearest You!
EAST COAST 615-478-5760
WEST COAST 206-882-2009
MasterCard VISA

SPOC
Your Amiga Becomes SPOC — Special Programs & Operations Computer! Your family can interact and compete with SPOC in over 35 highly entertaining ways. Contains educational and demo sections to show off your Amiga. Your family will run this disk more than all your other entertainment software combined or your money back.
You also get a disk from SPOCPD, our best of public domain —great programs, reworked to run smoothly and easily. Both disks come with plain label and compact code — you do not pay for copy protection and fancy packaging. Why pay $30.00-$60.00 for one program when you can obtain a complete entertainment library for only $25.00 to:
SPOC
Box 299
Kiowa, OK 74553

DR. OXIDE SLICES PRICES!

Comp-U-Save's cost-cutting clinician is at it again! This month Dr. Oxide, a bargain-basement surgeon if ever there was one, offers the industry's lowest prices on hardware and software, plus special deals on products available only through Comp-U-Save!

BUILD YOUR OWN HARD DRIVE KITS!

5¼" powered hard drive chassis with fan
3½" powered hard drive chassis with fan
only $120.00 each

1000 or 500 SCSI interfaces with passthru and software for

$145.00

All you need is a SCSI hard drive and you're ready to Rock and Roll!

Call for SCSI hard drive prices

AMIGA PUBLIC DOMAIN

OVER 700 DISKS!

Largest Amiga PD Library in the World

also C-64 & C-128 – Write for Free Catalogue

Amiga PD – $4.00 each

EXTERNAL SLOT EXPANSION FOR AMIGA 500 and 1000

Now you can use most of the expansion cards designed for the 2000-hard disk controllers, 2/4/8 meg RAM cards, A2088 Bridgeboard, etc. Use low cost IBM compatible expansion cards already supporting a wide range of business and scientific applications. Designed to work with auto configurable cards. Meets Zorro Bus and Amiga 2000 Bus electrical specifications. Available exclusively through Comp-U-Save!

A500 2 SLOTS — $188.00
A1000 2 SLOTS — $188.00
A1000 3 SLOTS — $249.99

Amiga Hard Drives 500 - 1000 - 2000

20 Meg - $5.85.00 32 Meg - $699.99 48 Meg - $799.99

Amiga External Drive $139.99

Only Uses Half the Power of 1010 - with Pass thru

Panasonic WV1410 Video Camera

For Digitizers - $199.99 16 MM Lens - $29.99

Special 2400 Baud Modem - $149.99

★ Trackball	$45.00
★ Plastic Diskbank (Holds 120 3.5 in. Disks)	$14.99
★ Copy Arm (Heavy Duty)	$29.99
★ Mouse Mat (Fabric)	$ 5.00
★ Mouse Mat (Teflon)	$11.00
★ Gender Changers – All Types	Call
★ Static Mat (23.5 x 25.5 in.)	$24.00
★ Rapid Fire Joysticks	$12.00
★ Printer Buffer (32K–512K)	Call
★ RF Modulator (A1000)	$14.99
★ A/B Switch (Ser.)	$13.99
★ A/B Switch (Par.)	$14.99
★ A/B/C/D/ Switch	$29.99
★ Crossover Box	$39.99
★ Cables 500 – 1000 – 2000	Call
★ Teak Diskbank (holds 150 3.5 in. disks)	$39.99
★ Teak Diskbank (holds 200 5.25 in. disks)	$39.99
★ The Library (holds 80 3.5 in. disks)	$19.99
★ Floppy Wallets (Many Sizes)	Call
★ 3.5 in DS/DD Disks (Bulk)	$1.25 ea.
★ 5.25 in. DS/DD Disks (Bulk)	$ .39 ea.
★ Books — All Titles 20% Off	Call
★ Memory 512K — 8 Megs	Call

We carry tons of hardware and software for the Amiga. Call for prices.

Comp-U-Save

410 Maple Avenue, Westbury, NY 11590

In NY State (516) 997-6707 (Tech Support) Outside NY State (800) 356-9997 (Orders Only) Fax (516)334-3091

Come See Dr. Oxide
In Our Booth At All
World Of Commodore Shows!

The *TransAmi* DNet Network

A two-Amiga production system

by Chris Zamara

Whether you've read Matt Dillon's technical discourse on his DNet Network (sorry, "Link") or not, this article might be of interest to anyone who has - or requires - a two-Amiga environment. At the Editorial offices of *Transactor* and *TransAmi* magazines, we have a successful two-machine Link running, giving us full access of each machine from the other, file transfer, message-sending and mail capabilities, and a print queue. This was implemented using DNet and a hodgepodge of simple programs, shell aliases and hacks to DNet 'client' programs. We also use the network over a modem, allowing complete access to the Amiga at the office from home. This article outlines the basics of this ad-hoc network; the required programs will be freely available on the disk for this issue. The best aspect of this system, beside the fact that it is robust and highly usable, is that it's *cheap*; the hardware is a simple serial cable you can build or buy for a few bucks (we had one lying around), and the software is free.

When we first downloaded DNet from a local BBS and got it running (that makes it sound simpler than it was, but it makes us sound more competent, so we'll leave it at that), we were impressed by the ability to send files from one machine to the other without having to do anything special on the target machine other than have DNet running quietly in the background. We were also impressed by how robust it was: we could unplug the serial cable and plug it back in again in mid-send without losing data. At the worst, the send would be aborted, but the guru would seldom pop in for a nasty visit and spoil our fun. At first, this was all we could we could do with DNet: send a file or directory from one machine to another using the *putfiles* command. This in itself is a useful feature, and allows for all kinds of network-like capabilities.

Sending files

The *putfiles* program lets you send a file, or a complete directory, from your Amiga to the other. You specify the directory on the target machine that the file is to be copied to, and the file name on your machine. The copy will then commence, displaying the current number of bytes versus the total number as it proceeds. We run our network at 19200 baud, which is the fastest we could push it. At this rate, file transfer is reasonably quick. A typical *putfiles* command might be:

```
putfiles -dc: c:FileZap
```

This will copy the file *FileZap* in your C: directory to the C: directory on the other Amiga. Using *putfiles* directly in this way is one of the most-used features of the network. For example, the above command would be issued when the user on the other Amiga shouts across the room, "Hey, do you have a copy of *FileZap* over there?". "Yup... I'll send it to your C directory for you", you reply, as you punch in the command and save yet another 10-minute search for an unimportant floppy to use for file transfer duty. Surprisingly, we even found that both Amigas can send files to each other *at the same time!*

But while we could at first send to the other machine with the *putfiles* client program, we couldn't get *scli* - the program to run a remote shell on the other machine - to work properly. As it turns out, Matt wrote *scli* to work from a VAX or something, and not from another Amiga.

Remote shell

By modifying the *scli* C source and adding a shell alias and a few script files, we were able to get a real shell, with limited command line editing, that runs on the other Amiga. Here's an example: the two Amigas (called "Curly" and "Moe") are connected via DNet. (Sadly, "Larry" is left out until Matt comes up with a three-Amiga version of DNet.) I am currently using "Curly", and from the DOS shell I type **remote** (an alias that invokes our special version of *scli*). A window pops up near the top of the Workbench screen, and after a few seconds displays a familiar AmigaDOS shell prompt. This shell will in many ways behave exactly like a normal shell or CLI window, except that it is running on Moe, the other Amiga. In other words, when I type 'Dir', the directory I see is the current directory of that shell *on Moe*. If I run a program, it runs on Moe, so I can play tricks on whoever is using Moe at the time and make clocks pop up unexpectedly on his screen. At least one other *TransAmi* editor can verify this fact, although it has

not proven to be one of the more popular applications of the network.

Messages and mail

With a *send* command, a simple one-line message can be transmitted to the CLI or shell window on the other Amiga. For example, if I type 'send Look out! Your computer is smoking!' in my CLI, the screen on the other Amiga will flash ("DisplayBeep"), and the message will appear on the current CLI line. This may not sound very useful if the person is in the same office, but remember, the network works over the modem as well, so if I am working here at the office and hooked up via the network to Nick at home, we can send simple messages in this way. For longer communication, we can use DNet's terminal and have a window constantly up that we can type into, but *send* is usually preferable, since you don't need the extra window hanging around.

Send does not go about its business in perhaps the most elegant manner. We probably should have written our own DNet client to do the job, but failure in that endeavour instilled the "Let's just get the job done, dammit" work ethic, and the current form of *send* is the result. *Send* is simply a shell alias that creates a file in RAM containing the text to be sent, and does a *putfiles* to a special directory in the other Amiga's RAM. On that end, a program running in the background called the *NoteDemon* checks for this file every half second or so, and if it finds it, beeps and displays the contents on the CLI window. Primitive, but it works, and has no side effects of great consequence (the CPU time used by the demon checking for the file occasionally is negligible).

A more complicated system allows us to send mail to any number of other users on the network (although only two users can be on at any given time). A *logon* command is used to inform the network of your presence, and the *mail* command can be used to send a file (usually text in the form of a letter, but it can be anything) to any user. Mail can be sent local or remote; local mail is sent directly to the user's machine where he can use it at will, and remote mail goes in a special area on the other machine (the one you - the sender - are using), where the recipient can still get at it and view it with the remote shell, but has to transfer it with *putfiles* if he wants to read it locally. When you log on, a message informs you if mail is waiting for you, and whether that mail is local or remote. If mail is sent to you while you are logged on, the screen will flash and a message will appear on your CLI window informing you of this fact. You may log on to receive local mail, remote mail, or both.

The mail system consists of the CLI commands *mail* and *logon*, and the background task *MailDemon*. It is probably the most advanced network-like feature we have implemented with DNet, but it is also the least used, probably because nobody wants to go through the trouble of logging on every time they get behind the keyboard. Since the heady days of the initial implementation of mail, we have pretty much gone back to leaving messages on bits of paper, or using even more primitive communications systems like talking to each other.

The print queue

Because we have one laser printer - in constant use - and more than one Amiga, the print queue is what we have come to depend on most in the network. Although the printer is connected to Curly, you can output a file directly from Moe through the network, and both users can send files for printing at the same time. The print queue works in a very simple manner: The printer is connected to Curly via the parallel port, and another 'Demon' - the *PrintDemon* - is always hanging around in the backround on Curly, checking for files in the logical device "pq:", which is squirrelled away in some subdirectory in RAM. When the *PrintDemon* sees a file in pq:, it sends its contents out the parallel port to the printer, and deletes the file after it has been sent. All you have to do to print, then, is copy a file to that directory, and the demon will send it out when it gets around to it.

This simple scheme has powerful implications: first of all, you can print a file from the remote machine easily with *putfiles*, (and even more easily with the *queue* alias that we use). But just as important, you can submit print jobs (this sounds more like a mainframe system all the time) to the print queue at any time, whether something is being printed at the time or not. So we have a primitive kind of print spooler here: you can look at the jobs currently in the queue with 'dir pq:', you can delete a job from the queue, and you can "drain" the printer by killing the demon, temporarily stopping all print jobs from being output. You can also get two large jobs printed at the same time: if each user queues his output as a separate file for each page (it is a PostScript® printer, which is page-oriented), both jobs will come out interspersed with each other. What you *can't* do (yet) is change the priority of print jobs, set job classes, cancel a job currently being printed, and other fancy print spooler functions. The demon also has a tendency to inexplicably eat a print job once in a while, but for a five minute hack to the network (five minute hacks rarely take more than a few hours), the print queue is really quite a good deal.

Build your own T-DNet

What you've just read about is a system that evolved because there was a need for it, and it is a constant ally in the monthly struggle to get out yet another *TransAmi*. Because DNet is flexible and general-purpose, the system was quite easy to put together to suit out needs. The moral here is that for any application you might have involving two Amigas, you should be able to custom-tailor a system for yourself using DNet, and perhaps some of the ideas presented here. You might find that any one of the capabilities mentioned above - file transfer, message sending, remote CLI operation, or a print queue - would justify the effort of setting up such a system. With that, then, and *without* a step-by-step recipe for setting up your system, we hope you can get your Amigas on speaking terms with each other.

Whap!

A history from the tester's side

by Doug Winger

For those of you out there that have never used an information service, the following will not be too instructive. It will, however, give you an idea of what a beta tester goes through in his daily rounds. The intent of this article is to show you what beta testing is about, along with a bit of insight into using those information services.

First, some background and history. I first saw *Whap!* one cool fall day when I was visited by one of the system operators from CompuServe's Amigaforum. He had recently moved relatively nearby to me and, after some on-line conversation, decided to visit. He was known for his work in graphics on the Amiga and brought along some of his work. After some time of oohing and ahhing over his work in progress, he asked me if I could keep silent about a new program. I had my suspicions about what it was. They were borne out when he showed me what he had in his bag of tricks.

There had been a flurry of activity in Amigaforum, with the message base and conference area being filled with the subject of a possible *TapCIS* for the Amiga. *TapCIS* is a marvellous program for MS-DOS machines that automates on-line time, reducing the cost and easing the way for users of CompuServe. It is an outgrowth of *AutoSIG*, another MS-DOS program. It has enjoyed great success in the IBM world, and many of the Amiga owners who had Bridgeboards had used it and raved about it. Recently, CompuServe had announced *Navigator*, a similar program for the Mac. Many requests for something similar for the Amiga were soon seen in both the message base and in conferences. Questions on how such a program could be set up on the Amiga abounded.

Those who have used an information service have discovered that one of the most trying of times is the initial learning process. CompuServe is very easy to use, having menu choices for just about everything, and on-line help available everywhere. It also has one fault - it is expensive. One of the first lessons learned by the knowledgeable user of that service is that menus eat time and that high baud speed is expensive when you fumble around. Another lesson learned is that, while it is expensive, it is perhaps the most informative service available, with the greatest range of subjects and interests being available to the user.

The (then) Amigaforum had a membership of slightly over ten thousand people, ranging from the neophyte with no previous computer experience, to the expert commercial hardware and software developers. It was a lively place. The message base was receiving over two hundred posted messages a day. Keeping up with this flow of information was difficult. Replying or originating a message while on-line, let alone reading all the messages, tended to take a very long time. On-line times of half an hour to just read messages were common. And this was only one of many forums.

Methods to get around this were kludged together. By using an ASCII capture and the proper commands one could 'dump' all the new messages in sorted order. With some careful planning and the proper software these message raiding sessions could be kept quite short. By using an editor or file reader off-line, you saved the time you would normally spend doing the actual reading. Several specialized programs were seen, such as *KEEP*, that would allow you to go through them one by one, separating the messages of interest out from the rest. If one was knowledgeable about the commands needed to leave messages, a sort of script could be set up to reverse the process, allowing the user to 'dump' his replies to the base.

There were drawbacks to it. If a message you were replying to had been deleted while you were busily answering it off-line, and your script called for a reply to that message, things tended to blow up. The user was totally dependent upon his carefully thinking out the entire session beforehand. Many clever combinations of terminal software, utilities and editors were seen. Each one had drawbacks to it.

An integrated approach

TapCIS had got around this by incorporating the capture, edit, dump cycle into one program. It had the added advantage of some 'intelligence', handling the unexpected for the user. It also allowed one to make download scripts, to catalogue the

Data Libraries and download the individual files chosen by the user. All this was done automatically and had the effect of drastically shortening on-line time. CompuServe was thrilled with the results of the use of TapCIS, and came out with their Mac version, *Navigator.*

"CompuServe thrilled? They lose money and are thrilled?" you say. They studied the phenomenon when it first started out, and discovered something very interesting. Usage went up. Individuals who had previously visited only rarely started logging on more often. The new user no longer had to learn the details to get effective use out of the service. They were getting more for their money and could now use the service more effectively and at lower cost to themselves. People who had previously visited perhaps once a month started logging on weekly, or even daily. The message base, previously used only by the expert or the desperate, now was being used by almost everyone. The confusion experienced while first learning to download files and make use of the data libraries was eased. Everyone was happy. Except for the Amiga users.

Amiga users get happy

My suspicions were confirmed. *Whap!* was revealed to me. It was then in alpha testing. I was only shown the main screen and the editor at this time, but it looked good. The front end was filled with all sorts of gadgets, while the editor was absolutely lovely. 'Push button' gadgets, a great colour scheme, and easy use made it truly a nice looking package. I received the nickel tour of its use and was duly impressed. The author, Jim Stevens, had done a very nice job.

Several days later, I was asked by the forum administrator, Steve Ahlstrom, if I would be interested in being a beta tester for it. I hurriedly yelled, "Yes!" and received an E-mail later that night stating the conditions I would have to observe while testing it: Non-disclosure under penalty of losing my first born child. Timely reports of any bugs found or suffer being deemed terminally slow. I replied, "Yes!" once again, and was soon ushered into the world of the beta tester.

Amigaforum at that time maintained a restricted access data library and message section, known as the VIP Suite. It was used as both a test-bed for new communications programs, and a non-public area for messages concerning them. I was eventually permitted access to it after it was cleaned up for use by the testers. On my first visit I found WHAP.ARC. After downloading and de-arcing it, I printed out the incomplete but thorough documentation file. I carefully followed the instructions for installation, and fired it up for the first time. I immediately gurued. Another try after rechecking everything resulted in yet another guru. A third time, another guru.

I went through a series of install-run-guru cycles, trying every variation I could come up with. Desperate, I left a message reporting my travails manually. While waiting hopefully for a reply, I once again went through the documents. Still no go. The message in reply to mine requested more details, asking for a detailed description of *exactly* what I had done. As I replied, I found what had gone wrong. I had copied the supplied font to the proper directory, making the correctly named directory for it but had copied the name as it had popped out of the archive. Font sizes of whapcis8 do not work too well. I had just received my first lesson as a beta tester: *Keep track of what you're doing at all times, and be prepared to give an accurate description of the steps you have taken previous to the failure.*

With that out of the way, *Whap!* was a revelation. It would also guru quite colourfully for many reasons. It had a wonderful front end to it, allowing you to pick your forum and whether you wished to capture messages, send an original message, create a data library catalogue, or download a file from a previous catalogue. The combination message-reader/editor worked well, as did the terminal included in it. Most times, that is. If it couldn't find the expected font, guru. If it couldn't find the expected devices, guru. If you looked at it too closely, guru. But, wonder of wonders, it worked! It would go into the forums you had set and capture the messages in the sections you had set. It would allow you to read them off-line, and reply to them. It allowed you to originate your own messages, starting a new thread of messages if you wished. It would catalogue, and then download the files you selected. It would also guru if it decided it didn't like the way you typed.

The *Whap!*/CompuServe environment

CompuServe allows the user to set up his environment on-line, allowing several different terminal settings, whether or not to use menus, which service you go to upon first logging on, and even setting up a personal menu of choices. *Whap!* in its first incarnation required that certain of these settings be done beforehand. Luckily, I had been using all of these options regularly, so resetting them to suit *Whap!* was no problem for me. Other testers were not so fortunate. *Whap!* was designed to use a seven-wire modem communications protocol, so that it could monitor the DTR line, keeping track of time spent on-line accurately and allowing recovery from a dropped carrier. Several of the testers did not have the capability of using it. One tester was found to have an acoustically coupled modem (!) and would manually dial up, and then place the handset in the modem's cradle. Others had only three-wire, or couldn't determine how to set their modems for DTR. The first of many blocks of messages were posted in message section 14.

Whap! has a parameters window, which allows you to set up: your name, telephone number to dial, secondary and tertiary numbers to dial if the first number doesn't get through, the baud rate to use, modem control strings, your ID number, and your password. The documents that were included with the release were thorough, if a bit vague on the finer points. Default settings were supplied, but the number of different modems and methods of hook-up, and the differing networks used by the testers soon had the message section filled with cries for aid. Within a week, the grosser bugs were eliminated in a revised release. Requesters appeared, warning of *Whap!*'s

inability to find something rather than depending on the Guru to chastise the user. A more flexible parameters window was provided, allowing for the use of three-wire modem communication. The forum message section selection was revamped. It was in the process of becoming civilized.

Beta testing by the book

It was during this time that I worked out a method of logging my usage. I would open a simple ED window in the background, allowing for quick notes to be made while using *Whap!*. I would note the time and what I was attempting to do, then start the session. Afterwards, provided the session didn't end with *Whap!* imitating a smoking crater, I would note the time, what errors (if any) had occurred, whether the captured files were readable and how the various options behaved. I would reserve 'special' sessions, where I would push buttons at random and see what would happen. I would try to make it crash, and often succeeded. Each bug was duly noted and, provided some other tester had not already posted similar findings, they would be relayed to Jim Stevens in Section 14. I soon found that keeping notes with this method had drawbacks. A serious guru would wreak havoc, destroying my carefully written notes. (I was using VD0:, ASDG's RRD, to be safe, but some errors would trash the system completely.)

Another responsibility of testers is to make suggestions. Changes in method or desired additions were requested, and were soon flowing in. Each release had something new in it. If a suggestion was worthy or practical, it was promptly incorporated into the next release. If not practical, the author would duly thank the tester, and either promise to incorporate it in a later version, or give the tester sound reasons why it would not be. Things were coming together, and then we got a surprise.

***Whap!'s* evolution**

The original version required at least one megabyte of memory. It incorporated the actual retrieval mechanism, the reader-editor, and the terminal in one program. The first major variation was released, and was an instant success. The program was divided into two separate modules, and the slightly klunky terminal was dropped. *Whap!* had been divided into *Whap!* and *View!*, and allowed you to use the terminal of choice for interactive use. You could now use *Whap!* as an autodialling front end, or drop out of *Whap!* into your terminal while on-line. It also allowed it to run in a 512K machine. The basic bugs had been worked out, with some annoyances still needing to be tracked down and eliminated.

It was at this time that one major debate about options began. One of the commands used by many users of the message base on CompuServe was QSN. This command would scan the message headers of new messages, arranged by subject, and note how many replies there were to the original message. It was a time saver, as the user could capture the much shorter scan, and select which thread to read later. The problem was that most testers read their sections in their entirety; to them, QSN seems like a waste of time, requiring two on-line sessions to actually read messages of interest. Arguments both pro and con flew, and soon another was added.

Normally, messages addressed to an individual could be retrieved with a separate command to read marked messages. *Whap!* did this automatically, retrieving them from all sections, ensuring that they were received by the user. The complaint was that this would mean that the marked messages were read twice, once during the RM (read marked), and again when an RTN (read new in thread order), was issued. This may seem like a non-issue, but the arguments on both sides were hot and heavy at the time.

Pleas from testers began to overtake bug reports. New versions were coming out regularly, being both bug fixes and adding new options. *Whap!* was slowly going from somthing to be tested to something that was being used. Message files could be appended or overwritten, being divided into sessions. Catalogue files were now easier to use, and contained additional information. Preparations for future options were seen, and information entry windows were tweaked. Various goodies were added, like the ability to make a User ID listing from messages. The Quick B protocol transfer was cleaned up, and made easier to use. Bugs were dying left and right, but there were some that seemed invulnerable. And then CompuServe announced they were introducing new software.

Once you learn the rules, they change the game

The new software changed many commands. Thankfully, there was a method to get around them and use the old. *Whap!* was slightly revised, and went on from there. It was at this time that I 'met' the author. I had recently changed network nodes (where you connect to the network) and ran into a problem. *Whap!* would not log on through it. The delay between the modem making connection and *Whap!* sending the first of the log-on strings was too short. I was given a number to call, and was soon speaking to the author. After some raving on my part about the program, a new version was soon in the works. (I could hear Jim typing in the background.) The next day it was available, and promptly downloaded.

Due to the nature of the program, and the intended use, many of the problems that most other software goes through while being tested were surmounted. Feedback to the author was almost instantaneous, and detailed instructions or necessary files could be gotten as fast as they could be transferred. Private talks in conference allowed for real-time interaction. Electronic mail through Easyplex allowed for private, individual attention. One could upload bug files to the author, bypassing the need for the tester to first determine what went wrong, and allowing the author to see results of something for himself. Fixes and notices could be posted as they were available, and variations could be sent to individuals for testing.

Whap! was now very usable, but some bugs remained. Amigaforum was due to be split soon, and *Whap!* seemed to go in-

to hibernation. Bug reports still trickled in, but were negligible; suggestions for improvement or change were the rule. The testers knew that the next version would be the first real release.

Amigaforum split, and was now divided into three separate forums. AmigaTech would be the closest to the old Amigaforum, containing the tools and utilities, and the programming sections. AmigaArts would get the graphic and music files and utilities, and contain the application software. AmigaVendor would be for on-line vendor support, allowing for user feedback and the answering of customer questions for several companies. And *Whap!* was released.

Whap! 1.0 went out to the public almost fully intact. The only change was the addition of 'nag screens' to the regular version. Testers were allowed to download the non-nag version. The forum administrator had opened a library and message section devoted to *Whap!* and its use. Downloads started almost immediately. And there were suddenly a whole new set of bugs and problems revealed.

"No plan of battle survives contact with the enemy." That saying had certainly proved true in this case. Some of the bugs found had been in *Whap!* since the beginning, and had never been completely eliminated. New bugs and problems were revealed. *Whap! 1.1* went up very soon afterwards, followed sometime later by 1.2. Some things were taken out, such as the auto-start feature that allowed *Whap!* to wake up at some preset time and go do its thing. One user discovered to his dismay that *Whap!* had hung, and remained on-line for several hours. The author quickly removed that. Bugs that had been noticeable in early versions, such as *Whap!* losing its bearings due to line noise, soon were happening on a large scale. Timeouts due to system traffic became very commonplace, as the forum split had temporarily forced the Amigaforums onto a very busy host computer. *Whap! 1.3* was soon out, followed shortly by the version current at the time of this writing, 1.4.

Version 1.4 was a major change, with the front end changing quite a bit. Additional filtering helped greatly with line noise, and time-out values and recovery were made much more robust. Editor changes were few but welcome. They also introduced some bugs of their own. Many of the previous suggestions made during testing, but put off until it became practical to implement them, were now in place.

Whap! also introduced a novel method of registration in its release versions. Upon receipt of the $39.95 US asked by the author, a special binary E-mail would be sent to the user, patching the program to allow use of some additional options, and eliminating the nag screens. My final bit of testing of *Whap!* occurred while receiving the registration. I had recently taken to using Workbench to start *Whap!*, and the one major bug left was discovered this way. Registration would not take place, nor would the necessary stack setting be done from it. CLI use would not show these problems, but Workbench did. A patch to 1.4 was released almost immediately. *Whap!* still had bugs, but they were now very, very small. It now worked very well. Workbench start-up was now in place, as was stack setting. A very thorough documentation file was included in the release, as were installation scripts and the ability to use alternate networks to enter CompuServe. *Whap!* now would completely set on-line parameters automatically for the user, reducing first-use problems.

Whap! now allows the user to retrieve messages with QSN or to read entire sections. It will make catalogues of the data libraries, and automatically download files matching patterns set by the user, or selected from the catalogue later. It will transfer binary files through E-mail. It will allow for storage of messages either as a permanent file (appending new messages as they are received), or retaining the last session as back-up and deleting earlier sessions. One can reply to any message, or originate messages, and they will be posted automatically. Back-ups can be made for almost everything at the user's discretion. Network foul-ups are handled for the user, with *Whap!* showing almost human responses. Noise handling is still a bugaboo, but *Whap!* will now handle almost all of it, and implements a deadman timer. The user's billing rate can be set, and it will display a graphic representation, or a more detailed breakdown by forum, with estimates for future billing.

The editor is easy to use (but does have one annoying bug in it, easily avoided) and the ability to set where various files are stored is very convenient. High baud rates are now economical for the user, as there is no wasted time. In fact, a saving by using them is one of the important features of *Whap!*. Compuserve's basic rate of $6.00 an hour for 300-450 baud is steep, as is the price of $12.00 an hour for 1200+ baud. But some simple arithmetic reveals an interesting fact. At 1200 baud, you're paying double for the time, but getting four times the information. At 2400 baud you're getting *eight* times the information. With *Whap!* doing the driving there is no wasted time, and the efficiency of *Whap!* results in a reduction in the time spent actually retrieving that information, which, in turn, lowers your bill. A savings all around.

It should be noted that *Whap!* is still being revised, with some changes at CompuServe opening up some startling possibilities in the near future. A future version is already in the planning stages, and will take advantage of the unique capabilities that CompuServe is planning to implement. Aside from the addition of B-Plus (a new transfer protocol), these changes promise to obsolete many of the current methods in use. Consequently, use of CompuServe will become both less expensive and incredibly powerful. It will also mean a change in name, according to the author, to something a little more fitting.

I have watched *Whap!* slowly evolve over the course of several months, from something which consistently blew up to something robust and considered more than a good bargain at the asking price. for those of you who do use CompuServe, you can find *Whap!* in LIB 16 of AmigaTech, and will find that the author, Jim Stevens (PPN 72230,323), is available to receive your comments or answer any questions you may have.

Getting Mass Storage Under Control

A beginner's guide to directory management

by Dan Schein
Copyright ©1989 Dan Schein

Mass storage is one of those terms that has a different meaning depending who you ask. If you ask a new Amiga owner who upgraded from a C64 or C128, mass storage may be the Amiga's 3.5" 880K floppies. But if you ask a floppy based Amiga user, mass storage might be a 10 or 20MB hard drive. And if you ask a hard drive equipped Amiga user... Well you get the idea, but regardless of your definition of mass storage, this article is going to teach you how to organize it and why.

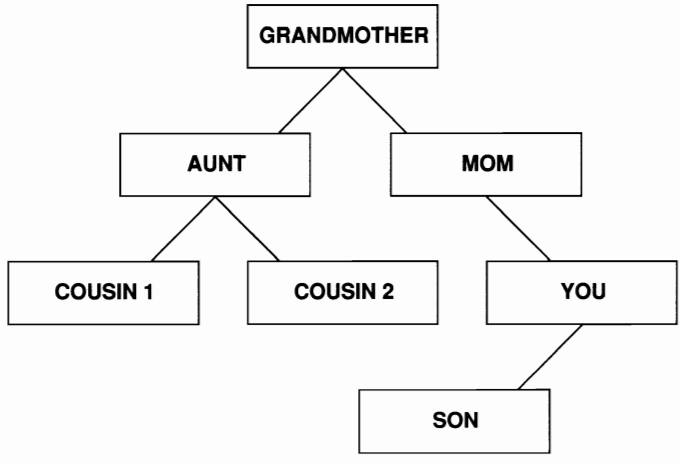
Directories and subdirectories; why?

To organize any form of mass storage, directories and subdirectories are required. If your background is from a non-hierarchical directory system (or if this is your first computer) then the idea of directories and subdirectories may seem confusing at first. Let's examine the importance of directories and their subdirectories. Keep in mind that directories become more important the larger your mass storage becomes. For example if you had a Commodore 64, with its 1541 drive with 170K of storage, it's easy to keep all your files in one directory (not to mention there is no other way to do it). Odds are that a 1541 will only have about 20-30 files on a disk, and looking at the directory to find a specific file is not hard. Now consider that Commodore made a D9090 7.5MB hard drive (for their PET/CBM line of computers) that kept all of the files in one directory. To make matters worse, it had an unlimited number of directory entries. Now imagine the mess when doing a directory of 500 (or more) files, not to mention the amount of time it took.

Back to the Amiga, let's say you have a collection of one week's messages from the *UseNet* newsgroup *comp.sys.amiga*, roughly 4000+. Each of these messages is stored in its own file; now add another 1500+ messages from a sister newsgroup, *comp.sys.amiga.tech*. Place these 5500+ file names all in one directory. No big deal you say, it's only a long directory. Well, that is not 100% correct; what would happen when a news article from one group happened to have the same name as an article in the other newsgroup? Well, on the Amiga the original is overwritten (deleted): no warning, no message, simply gone. This would not have happened if the files for each newsgroup were kept in their own directory.

Hierarchical file system? Huh?

A hierarchical structure is most often compared to an upside-down tree. Perhaps a better example might be comparing it to a family tree. If we trace a family's lineage you will see that a couple has a child, that child has several children, and each of those children can have more children. Somewhere in all of this you fit in, and somehow everyone is related to you. This is shown in the figure below.

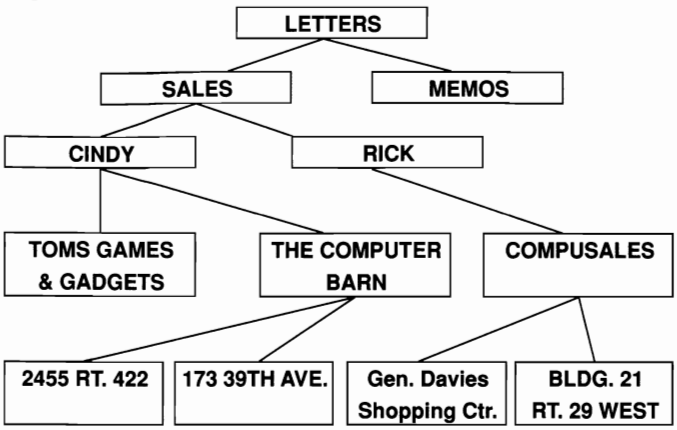


AmigaDOS and its hierarchical file system resemble this family tree and that is why the AmigaDOS file structure is also commonly referred to as a 'tree'. It is built out of a series of connected directories that form the tree. This type of structure allows you to organize your files so that you can easily find any particular file or directory. In AmigaDOS, an empty disk starts with one directory; from this directory you can make as many subdirectories as you wish. Each of these subdirectories may have as many subdirectories as you wish. In this fashion you can expand the structure (tree) to as many levels as you need.

A good idea is to dedicate each subdirectory to a single project (theme, subject, type, etc). This way the project will decide if more subdirectories are needed or not. An example is the president of a small company that sells Amigas (what else?). The president has a subdirectory called 'letters', and in that directory are two more directories, 'sales', and 'memos'. The

'sales' directory contains a subdirectory for each sales agent. For each customer an agent has, a subdirectory is created. And if that customer has more than one location, additional subdirectories are created. This makes the chance of important information getting mixed up almost none, and it also makes finding data on a specific agent or customer easy. An example of this can be seen in Figure 2.

Figure 2



If all this still seems a bit odd or hard to understand, then compare the mass storage device to a filing cabinet. Think of a directory as being a drawer in the filing cabinet and think of subdirectories as file holders. You could just throw all your bills into the file cabinet, but it would be a lot easier if you put the bills in one drawer, receipts in another, and so on. But if you were to organize that bill drawer with file holders, you could have a file (directory) for computer bills, utility bills, automobile bills, etc. Then if we really wanted to become organized, we could break up the file folders with more folders. One for electric, water, natural gas, cable, etc.

Terminology

Comparing the Amiga's file structure to a family tree is useful in teaching the parts of the file structure. The file structure and family tree are usually pictured upside down, with their *root* at the top. Our tree should be thought of as growing downward, that is having branches coming from the root. The root is always at the top of the tree and all branches grow down from there. Down is away from the root and up is towards the root. Branches can have other branches and they may also have leaves. Branches should be thought of as *directories* and leaves should be thought of as *files*. Figure 3 shows the same tree structure as Figure 1, with names added for our study. Notice that some branches have two names, subdirectory and directory. To the directory above (the parent), this branch is known as a subdirectory (just as I am known as son to my father). But to the directory below (the child) this branch, is known as a directory (just as I am known as father to my daughter).

Keep in mind that in addition to subdirectories, directories can also contain *files*. The root directory may contain files and/or directories just like any other directory, but it can not be a subdirectory to anyone - it will always be the root. A directory can exist and not contain any subdirectories or files; they are not required for a directory to exist. Also note that files cannot have directories or other files.

What's in a name?

In the following section, all information on naming applies to both files and directories, even though I'll use the term 'file'.

AmigaDOS allows directory and file names to be up to 30 characters long and contain any printing character except slash (/) and colon (:). This may sound a bit odd at first, using double quotes (") or a space as part of a file name, and you may have tried this already but could not get it to work. It does work and it can allow for some rather unusual names. There are two tricks (?) to getting AmigaDOS to accept a few of these characters. The first thing to remember is that a space, equals sign (=), plus sign (+), and double quote (") are recognized by the CLI. This means that if you want to use any of these in your filename, you must enclose the name in double quotes. The second thing to remember is that the double quote (") and asterisk (*) require an asterisk immediately before them. For example if you wanted your file to be called **DAN="yea" +23*** you must enter: **"DAN=*"yea*" +23**"**

In general, though, you should avoid using these special characters in your file names to avoid problems when using them in CLI commands.

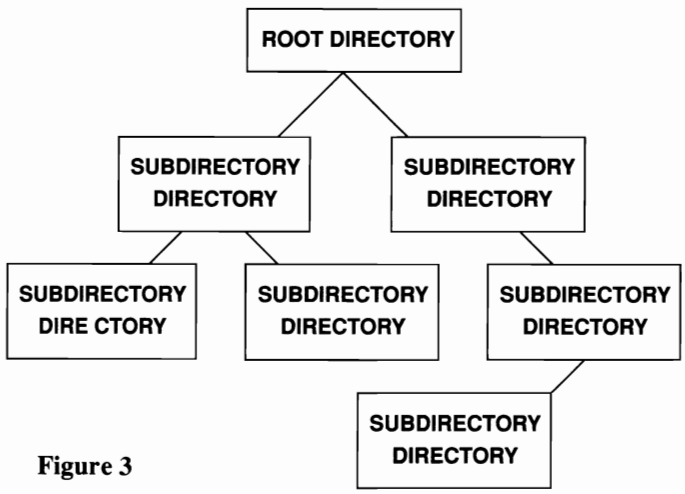


Figure 3

With that out of the way, let's discuss names. Like two children of one parent, no two files in a directory can have the same name. (This also applies to a file and subdirectory having the same name in one directory.) Just as children of different parents can have the same name, so can files of different parent directories. Try to choose names that will mean something to you at a later date. Far too often, directories are full of names that are totally meaningless a few days after they were created. AmigaDOS is not case-sensitive and considers Dan, DAN, and dan as all being the same name. Examples of good names are:

```
heidi_birth
transactor_report
uucp_offer
```

Another area that helps make filenames more readable and meaningful are *filename extensions*. An example of filename extensions as used in C programming are:

```
foobar.c   (the 'C' source code file)
foobar.o   (the object module)
foobar     (the executable program)
```

Filename extensions can be used for many reasons and should be used often. An example of using extensions for a magazine article (such as this one) might be:

```
article_part.1    (the first part)
article_part.2    (the second part)
article.1         (parts 1 & 2 joined)
article.1.2       (second draft, spell checked)
```

Other examples of extensions you might use are '.pic' for IFF picture files, '.wp' for *WordPerfect* documents, '.pp' for *Professional Page* documents, '.ps' for PostScript files, and so on.

Setting and moving about directories

I've explained why directories (and subdirectories) are needed, how the Amiga's hierarchical directory structure works, and how to choose names. Now let's put it to work (if you don't know how to create a directory, refer to your Amiga manual). Three important things to remember in moving about directories: First, the colon (:) always means the root directory. Second, the forward slash (/) will move you back (up) one directory (closer to the root directory of your current directory). Third, AmigaDOS always maintains a record of your current directory. Using Figure 1, let's set the root directory to be our Current Directory (CD):

```
1> cd :
```

We are now in the root directory of our disk. If we want to go to the directory "grandson", we would enter:

```
1> cd mom/you/grandson
```

To move back (up) one directory to "son", enter:

```
1> cd /
```

You could have also entered:

```
1> cd :mom/you/son
```

To move to the directory "cousin1", you could enter:

```
1> cd //aunt/cousin1
```

Or you could have entered:

```
1> cd :aunt/cousin1
```

Remember that AmigaDOS maintains our current directory and unless the path we enter has a colon (root) as its first character, the directory change is based on our current directory. As you can see, it is rather easy to move around the directories on an Amiga. One other thing you may have noticed is that directory paths can get rather lengthy. To help with this length problem, AmigaDOS has the ability to *assign* a name to a directory. If we want to refer to the directory ":mom/you/son/grandson" as "shorty:", for example, all we need to enter is:

```
1> ASSIGN shorty: DF0:mom/you/son/grandson
```

We can now use 'shorty:' as a shortcut and save ourselves a lot of typing and the chance of making a typographical error. This brings up another feature of the Amiga: device and volume names.

Device and volume names

Device names are the names AmigaDOS knows physical devices by. The most common example is the internal disk drive found in all Amigas. It is known as 'DF0:' to AmigaDOS. As you can see, it is a physical device and regardless of what disk is inserted (or if no disk is present), it can be accessed via its device name. Other examples of device names would be hard drives, RAM disks, printers, etc. Volume names, on the other hand, can refer to a disk (floppy, hard, RAM) or a partition of a disk (many hard drive users have several partitions). For example, if you bought a game on two disks, disk one might have a volume name of "Game_1", and disk 2 may be "Game_2". To check volume names you can use the AmigaDOS INFO command, or look at the disk icons on the Workbench.

Let's say that the disk with our family tree structure (Figure 1) is named "Family". We could have assigned 'shorty' in this manner:

```
1> ASSIGN shorty: Family:mom/you/son/grandson
```

No matter which method we used to assign 'shorty', when our family tree disk is not in a drive and we try to CD to 'shorty:', a requester asking us to insert 'Family' appears. We can insert the proper disk and continue (or cancel if we wish). One of the best things about this type of assign is that we can place our family tree disk in *any* drive and AmigaDOS will CD us to that disk. If our disk is not in a drive, AmigaDOS will give us a chance to insert it. Along with these advantages, another is not having to remember long path names.

Where to keep what, and why

One of the most important things you can do (and which is most often not done) is to isolate your software. What I mean

is keeping the software supplied by Commodore in a different directory than the things you downloaded from a BBS. Also, it's just as important to isolate software you buy.

The reasons for all this isolation are to make the tracking down of any software related problems easier, and to help avoid the accidental overwriting or deleting of existing files. A third reason is to keep your hard drive well organized. If you've ever kept a floppy around to be used as a 'download' disk, you know how hard it can be to try and find things weeks later. Multiply this by twenty or more and you can see a major mess developing.

Suggestions for directories are the standard ones found on the Workbench and Extras disk. I recommend you simply do a 'copy all' to transfer them onto a freshly-formatted hard drive. Next you will want a directory similar to "MY_C", this is where you store all 'C:' type commands that have not come from Commodore. Other examples are "Terminals" for your modem programs. Have fun, use your imagination and keep things isolated.

The end

This article is meant to be a quick introduction to Mass Storage. If you need or want more information, feel free to contact me with your questions and suggestions. I can be reached on *Berks Amiga BBS* at 215/678-7691 or via *UseNet* "sneakers@heimat".

Amiga

500 Computer, 501 512K Expansion, 1084S Stereo Colour Monitor	$1000
1010 3½" external	$159.00
2000 Computer	$1489.00
2000 HD Computer	$2199.00
2010 3½" internal	$160.00
2058 8MB Expansion	$625.00
2088 Bridgeboard w/5¼" drive	$499.00

Epson

LX-800	$199.00
FX-850	$359.00
FX-1050	$469.00
EX-800	$379.00
LQ-500	$338.00
LQ-850	$539.00
LQ-950	$609.00
LQ-1050	$759.00
LQ-2550	$949.00

Panasonic

KX-P1080i	$159.00	KX-P1592	$399.00
KX-P1091i	$199.00	KX-P1595	$439.00
KX-P1092i	$319.00	KX-P3131	$310.00
KX-P1524	$539.00	KX-P3151	$479.00

Packard Bell

1200B $79.00 2400B $149.00

Hayes compatible external modem

PC Plus

918-337-0266

Everything is in stock, ship the same day

Amiga is the registered trademark of Commodore Business Machines

Authorized Dealer for Amiga, Epson, Panasonic

New Store!

Fisher's

Computers & Software

Authorized Amiga Dealer

(714) 987-1662

Serving The Inland Impire

NAG® PLUS 3.0

Schedule Assistant

Software for the AMIGA™ computer

Voice and Sound Reminders — *Create your own or use NAG Plus library.*
Perpetual Calendar — *Enter 99 events per day.*
Auto Dialer — *Connects YOU, not your modem.*
AREXX Port — *Commands any timed event or action.*
Notepad — *Click on any word to open text editor.*
Print — *One click prints any event or appointment list.*
Suggested Retail $79.95. Ask your dealer or contact:

Gramma software For Your Own Good

17730 15th Avenue N.E.
Suite 223
Seattle, Washington 98155
(206) 363-6417

✓✓✓✓ UEDIT ✓✓✓✓

Programmable Editor
For Technical Amiga Users

"Definitely the most sophisticated text editor to come along last year . . . easy for beginners . . . and advanced users . . . reviewing UEDIT is like an art critic reviewing a lump of clay. You can mold it to suit your needs . . ."

Commodore August 1988

PURCHASING OPTIONS (Choose one. Foreign add $3.)
A) $47.00 + $3 shipping, **Uedit** with printed user reference manual
B) $44.00, **Uedit** with user reference manual on disk.
C) $25.00, **Uedit** with command language doc on disk.

Uedit has quietly become the editor of choice for bright programmers and professionals in the USA and more than a dozen countries. Recently it won the Gold Medal in Commodore magazine.

Features: Ultra power, capacity, and flexibility. **Uedit** has ARexx interface, many word processing features, optional spelling checker, and a truckload of editing features. How do you describe a program that lets you add new features when you feel like it? Yet you can use it right away "as is". Edit 20, even 100 or more files. Auto-load/edit/save a batch of files while you relax. Every command and feature is customizable, even the online Help and Teach Keys facility!

Rick Stiles, P.O. Box 666, Washington, IN 47501
(812) 254- 4986

Scripter

An easy screen script 'word processor'

by Jay A. Gross

Ah, desktop video. Ah, but before you take the lens cap off your camcorder, before the lights go on, before the action begins, you need something much more basic and down to earth. You need a script. Whip out your word processor to get your ideas onto paper, and presto! You've just discovered a major limitation of word processors: screen scripts.

For video production, a script is generally divided into two columns. The left column describes dialogue, action or what have you, and the right column describes video, background sound effects, etc. Consider this simple example:

SOUND	VIDEO
Sound: Thunder, downpouring rain	Pan right to empty armchair.
Thunder gets louder, more urgent Actor walks up to stand behind chair	Lightning flashes.
ACTOR: "It was a dark and stormy night!"	Lightning strikes set.

Fade to WHITE!

Getting those columns of stuff is not a problem with a simple typewriter. You just physically move the carriage to where the second part goes, and type away. To insert something, you roll the paper back. A problem, though, is that typewriters, unlike computer word processors, have no long term memory, even though some of the current ones memorize a line of text before they type it. With a typewriter, if you make a big boo-boo, you can't correct it on the screen, resave it to disk, and print another copy. You can't copy a block of stuff from one place and duplicate it in another, either, and to move a block, you'd have to retype at least a whole page. Besides that, if you make enough typos, you might want to invest in some White Out futures.

Even though they offer many advantages over even the fanciest typewriter, word processors on the Amiga and just about every other machine are mainly designed for single-column operation. You key your letter, article or novella into a single width of text shown on the screen. With a computer word processor, trying to do two-column screen scripts can be a frustrating experience, as the word processor insists on having tabs or spaces inserted, in order to put a line on the right side when there is no corresponding line on the left side.

Don't believe it? Type the tiny example screenscript into your word processor, and add a few words - any words - into the right side descriptions. Messes up the columnar formatting, doesn't it? Case rested.

A few word processors handle columnar matter. However, the Amiga's word processors that do so (WordPerfect and excellence!, for example) do it in a way that is not very useful for the specific task of screen scripting. Let's see somebody type a whole script into Pagesetter, or Professional Page, too. It would work, but the process would be rather painful. Even on those 'other' computers, which claim to have all the software in the known universe - *they* don't have many word processors that work in columns, either. Well, there are a couple of really nice ones - screenscript processors very specific to the purpose - but for what they cost, you can *produce* movies, instead of write them!

Oh well. So much for electronic marvels, huh? Put away the computer and get that old typewriter back from Great Aunt Martha. Not yet! Herewith, for you to use and enjoy, is a pair of AmigaBASIC programs that will help you cope on your Amiga with the complicated matter of screen scripting, and leave Great Aunt Martha to her historical novel of the Old South.

The programs

There isn't anything clever or unusual about the AmigaBASIC code, but it gets the job done. It works fast enough, too, that there's really no reason to bother compiling it.

There are three pieces to this puzzle: Hack-ED.BAS, SScript and Paster.BAS.

You'll need to use Hack-ED.BAS only once, in order to create some programs: two editing programs (which become the ScreenScripter 'Program Set') plus the DOS script file SScript.

You use Paster.BAS after you have your script completed; from then on you can do any further editing in the word processor or text editor of your choice. It welds the output of the two editing programs together, creating one composite file.

The key to this method is the simple line editor ED, which comes on your Amiga WorkBench disk for free. Well, it costs two thousand dollars, but you get a free Amiga with it, right?

ED lives with its more complicated sibling, EDIT, in the C subdirectory on your WorkBench disk. In a little less than twenty thousand bytes, this simple program provides rudimentary text editing capabilities, including block copy and move, screenwise and top and bottom of file cursor positioning, and line and block delete. It depends on Amiga Preferences to set up margins.

Of course, ED is a far cry from feature-laden ProWrite, excellence!, or WordPerfect, but it works, and it happily multitasks, too, even with itself! For the purpose of creating a screen-scripter, it offers the advantage of automatically saving a line-feed character at the end of each line of its files, which makes it easy to mate two of its files into one.

Two sides: two windows

To get the screen scripter, we'll open two copies of ED simultaneously. That's one for the left side of the script, and another one for the right side of the script. You can do this yourself from the CLI prompt, but if you use the handy Hack-ED.BAS program here, it will make two customized copies of ED for you (leaving the original one intact), so that the two programs appear in the correct places on your monitor automatically - you won't have to adjust window positions and sizing to get them to line up left and right.

In addition to producing these two special EDs, HackED.BAS will create a script file named SScript that you can use from the CLI to get them running easily.

(For reasons I do not understand, I have found that to have two EDs operating at one time, I have to RUN one of them, and 'do' the other. When I RUN both of them, I meet the Amiga's guru. Try it if you must, but be sure nothing valuable is going on in the background when you do.)

The script file lives in the S: subdirectory on your system disk. That's a drawer named "S" but, to the Amiga, it's also a software device, and it's *the* place for script files that will work with the CLI's EXECUTE command. In fact, if you don't specify a path, EXECUTE will automatically look in S: for any filename you give it.

When you execute the SScript script file, the Amiga will create two, side-by-side ED windows, conveniently labelled "LEFT" and "RIGHT". The left one will be creating (and overwriting!) a file named "LeftSide", and the right one will be doing its thing on a file named "RightSide".

Align the windows!

If you need to resize these windows for some reason, be sure to keep your text files aligned within the window. An easy way to do this is to type 'tick marks' into each window - aligned left and right - every screenful of copy. If you use an odd character, '@' for example, as the alignment tick mark, you can later search for them and delete, either before writing out the finished file, or afterwards, by putting the finished file into your word processor - as you'll want to do, anyway, for spell checking if nothing else. The trick is to keep the same number of 'ticks' in both windows, unless you search and delete them *after* the two sides are pasted together.

Using the two windows together is like using any other two Amiga programs at once. You click once with the left mouse button in the window you want to type into. To space the stuff downward on one side to match up with things on the other side, just hit RETURN as much as needed. This simple system is truly WYSIWYG (What You See Is What You Get) - unless you get the files out of sync (see above).

For you diehard touch typists who can't possibly bear to move your hands from the keyboard in order to use the mouse, the Amiga lets you move the mouse pointer from the keyboard by holding down the right Amiga key and the left or right cursor keys at the same time. Then, when the mouse pointer is positioned over the opposite window, hit the left Amiga and left ALT keys - just once, no need to hold them down. There! The other window activates, as though you had clicked the left mouse button. Just like magic, and at no time did your hands leave your wrists. This trick works on almost anything, not just ED.

The Hack-ED.BAS program is fairly verbose with its explanations of what's going on, so there's not much use in repeating that stuff here. To help prevent disasters, the program makes two simple checks to see that you're feeding it a proper ED program. First, it checks the file length, which must be 19,564 bytes. Second, it searches for a string of characters within the ED code itself that tells the Amiga where and how large to open ED's screen. It is this 23-byte string that gets changed in order to convince ED to appear where the BASIC program wants it to.

Hack-ED.BAS goes through the process of 'hacking' twice, adjusting the 23-byte string each time. If the program doesn't find the correct string, it accuses you of feeding it a non-standard ED program, closes all files, and ends.

These quick checks, however, are the only ones the program makes. It trusts you not to do anything uncool to defeat it. If you do, you're on your own for any 'results' you might get, and the guru meditation number will automatically be engraved on the inside of your monitor screen forever, to remind you of your misdeeds.

The Paster.BAS program looks for the filenames "LeftSide" and "RightSide" with a pathname you supply. If you do some-

thing wrong in the pathname, the program closes down without doing anything. Although the program reads data from the two 'side' files, it does not overwrite them or delete them. They will still be there when the pasting operation is complete.

IMPORTANT! Next time you EXECUTE the SScript script, it will happily overwrite (*Replace! Eliminate! Get rid of! Kaput!*) the two 'Side' files, and that's without asking you any questions and without announcing the disastrous news. If you wish to preserve these two files, you must rename them, using a CLI window. It takes two commands to do it. The commands are:

```
RENAME LeftSide <newfilename>
RENAME RightSide <anothernewfilename>
```

AmigaDOS will not let you rename to a filename that already exists.

Within Paster.BAS, there are warnings (with escape routes) that the output file will be overwritten if it exists. It will!

As a word processor, ED does have some limitations, to say the least. It isn't very friendly, and it ignores the mouse. There are no pull-down menus. On-line help is a big zero. You can't click where you want to go with the mouse button. It doesn't permit softstyles (bold face, underline, italics). Just to name a few of the biggies. Where else, though, could you get a simple word processor that can fit two of itself in less than 40K of memory?

Editing large files

Speaking of K's, the default size for an ED document is 40 kilobytes. However, you can convince it to accept more by adding a <SIZE> parameter to the command line. If you need to edit larger documents, add a numeric argument to the line in the SScript file for each side.

The argument must be numeric, without any punctuation ("60000" for example, not "60,000"). Examples:

```
LeftED LeftSide 60000
RightED RightSide 60000
```

The 'Side' files don't have to have the same size parameter, but they should, anyway. Also, if you insist on editing really huge files with even one copy of ED, the computer will slow to a crawl. The 40K default is a good maximum, as a matter of fact.

Within the Hack-ED.BAS program is a variable, appropriately commented, for changing the height of the windows created. Change it downward only. If you customarily work on the Amiga in hi-res (640 by 400), and you set this to a larger number for taller windows, you'll be on very friendly terms with the guru. ED has a hard time coping with windows larger than medium-res (640 by 200) screens permit. The author did not come by that piece of information by reading books.

Getting printouts

ED assumes that your work will be stored to a file, and doesn't even have a print option. However, you can use the Amiga's TYPE command to preview either side of the script on your printer. If you roll the paper back carefully, change Preferences' left margin to 40 (or whatever), and print the other side on the same sheet, you can even see how the whole thing will look. This method is more laborious, however, than simply running the Paster.BAS program and typing *its* output file to the printer.

To see a print-out, first save the files from both of the sides. You do that by hitting the ESC key and then typing the ED command SA RETURN. Do that once in *each* window. If you like, and if your system has plenty of memory, you can leave the two EDs running while you multitask back to CLI to see a print-out. Here's the command:

```
TYPE <filename> TO PRT:
```

Since left and right margins are an alien concept to ED, any printing you do will be aligned, paged, and bordered based on the way you have your Amiga Preferences configured. If you wish to bypass Preferences, use the appropriate one of the following commands, depending on whether your printer is connected to the parallel port (the usual) or the serial port:

```
TYPE <filename> TO PAR:

TYPE <filename> TO SER:
```

These commands will give you unformatted text, or text matching whatever the printer is set up on its own to do.

The dividing line

The default settings for left and right EDs put the imaginary dividing line in the center, at column 40. However, there may be times when you want the column divider somewhere else, especially if you're planning to have a bordered document on an 80-column printer.

If so, just give Paster.BAS the column number for where you want the joining to occur. You should also resize the windows on your Amiga screen, so you can tell what's going on, and so ED will automatically wordwrap for you while you're creating the original files.

Some screenscripting conventions have character dialogue go all the way across the page, while video and audio effects are described in columns. The double-ED method will work in this case, too, but you have to do a little bit of tomfoolery in the left window to create a long line. The **ESC J** command will join two lines into one, eliminating the line-end sequence. The window will then scroll right as you move the cursor on the longer line. Be sure to put corresponding RETURNs in the right-hand window to allow the correct amount of space.

Using ED

Check through your AmigaDOS manuals - one came with the machine, remember? - for information on using ED. For simple purposes, you can get by with knowing just a few commands, but if you need the more complicated ones, they're there for you in all of their unfriendliness.

Such as it is, ScreenScripter is, alas, the state of the art in Amiga screen scripting programs. However, with scripting at last at your fingertips, you can now take up where you left off with the World's Greatest Video Script.

Listing 1: Hack-ED.BAS

```
'  Hack-ED.BAS
'  by Jay Gross
'
'
'  AmigaBASIC program for creating LeftED and RightED editing programs
'      out of the AmigaDOS editor, ED, for use in screen scripting, and
'      for creating AmigaDOS script files in appropriate places to make
'      them work together.
'
'
'
CLS                                     '  Get the nudies off the screen
IF FRE(0)<40000& THEN CLEAR, 40000&     '  Open up some precious memory
ON ERROR GOTO Problems                  '  Look out!  It's an error trap!
'
Hello:
'
COLOR 3,0
PRINT "        Hack-ED.BAS by Jay Gross" '  Credit the creator of this mess
COLOR 1,0
PRINT " "
PRINT "   This program creates two specialized editing programs.  The source"
PRINT "   file is AmigaDOS ED, found in your C: directory.  In order for it"
PRINT "   to work, you must have ED in your SYS:C subdirectory, and you must"
PRINT "   have room for two copies of it (about 38-K) on your DESTINATION"
PRINT "   disk.  The original ED file will NOT be removed OR changed in any"
PRINT "   way, but just in case, do NOT use original disks.  Always use"
PRINT "   backups.  The program OVERWRITES existing files without asking."
PRINT "   Nuff said."
PRINT "   "
PRINT "   First, a little information please..."

GOSUB FileDef                           '  Get a path and such

COLOR 2,0
PRINT "   Okay, ready to go!  If you're ready, too, hit most any key"
COLOR 1,0
GOSUB Any                               '  Wait for a keypress

COLOR 2,0
PRINT "   Opening Files.  Looking for ED . . ."
COLOR 1,0

GOSUB FindED                            ' ED?  Are you there ED?

PRINT " "
PRINT "   Okay, Ed checks out. This won't take long."

GOSUB MakeEds                           ' Do the DEED!

PRINT "   Whew!  That's done.  Next we create a file that you can EXECUTE"
PRINT "   from the CLI to use these new EDs easily.  Hold on a sec while"
PRINT "   that proceeds.  This file will go into your SYS:S directory."
PRINT "   That's the current S: device/directory."

GOSUB Script                            ' Write a script to S:

PRINT "   Done!  And effortless, too!  Now you're ready to use the new EDs."
PRINT "   The command is:  EXECUTE S:SScript  . . .and away you go."
PRINT "   click the right-hand window when you want to type something on"
PRINT "   the right side of the page.  Click the left window to type"
PRINT "   in the left column.  Easy!"
PRINT " "
COLOR 2,0
PRINT "   When you're done, be sure to save your work from BOTH windows!"
PRINT "   You MUST do two separate SAVE operations.  Don't forget!"
COLOR 1,0
PRINT "   Click in each one and hit <ESC>, and then type:  SA <RETURN> "
PRINT " "
PRINT "   The last step is to RUN the Paster.BAS program to paste both"
PRINT "   sides together as a single document."

END                                     ' Terminate and go away

'
'  ============   GOSUBS start here

FileDef:
COLOR 3,0
PRINT "   Please supply a DESTINATION path for the two output ED files."
COLOR 1,0
PRINT "   That means any valid Amiga pathname."
PRINT "   Examples:  DF0:  DF1:  DF0:C   DF1:ScreenScripters"
PRINT "   The path you give must be a device or directory, and it must exist."
PRINT "   Do NOT add any filenames.  The program will supply those."
LINE INPUT "   Path =  ", FileDef$
IF LEN(FileDef$)<2 THEN
  PRINT "   Using default path (SYS:C), okay?"
  WHILE approve$ <> CHR$(27)                    '  Allow <ESC> to break in
    approve$ = UCASE$(INKEY$)
    IF approve$ = "Y" THEN
      PRINT "   Okay!  Here goes. . ."
      FileDef$ = "Sys:C"
      RETURN
    END IF
    IF approve$ = "N" THEN
      PRINT "   Try again?  Hit <ESC> to stop this program right here"
      PRINT "   if necessary."
      FileDef$ = ""
      GOTO FileDef
    END IF
    IF approve$ = CHR$(27) THEN
      GOTO Problems
    END IF
  WEND
ELSE
  COLOR 2,0
  PRINT "   Using path " FileDef$ ", okay?"
  COLOR 1,0
  WHILE approve$ <> CHR$(27)                 '  Allow <ESC> to break program
    approve$ = UCASE$(INKEY$)
    IF approve$ = "Y" THEN
      PRINT "   Okay!  Here goes..."
      RETURN
    END IF
    IF approve$ = "N" THEN
      PRINT "   No?  Try again, then.  Hit <ESC> to stop now,"
      PRINT "   if necessary."
      FileDef$ = ""
      GOTO FileDef
    END IF
    IF approve$ = CHR$(27) THEN
      GOTO Problems
    END IF
  WEND
END IF
RETURN


Script:
OPEN "S:SScript" FOR OUTPUT AS #4          ' Scripts should all live in S:
'
PRINT#4, ";   This command file is for use with LeftED and RightED."
PRINT#4, ";     EXECUTE this file to paste Leftside and Rightside"
PRINT#4, ";     text files into one text file."
PRINT#4, ";"
```

```
PRINT#4, "RUN LeftED Leftside"
PRINT#4, "RightED Rightside"
PRINT#4, "ECHO OKAY!"
RETURN


FindED:

OPEN "sys:c/ED" FOR INPUT AS #1                    ' Adjust path string
IF RIGHT$(FileDef$,1)<> ":" THEN
  FileDef$ = FileDef$ + "/"
END IF
LeftDef$ = FileDef$ + "LeftED"
OPEN LeftDef$ FOR OUTPUT AS #2
RightDef$ = FileDef$ + "RightED"
OPEN RightDef$ FOR OUTPUT AS #3
'
'   If these OPEN statements are unsuccesful, the ON ERROR GOTO
'   statement at the beginning of the program will CLOSE all files and
'   abort the program here.
'
PRINT "   File open successful.  Now let's see . . ."
PRINT "   A proper ED program has 19564 bytes.  Yours has " LOF(1) " bytes!"
IF LOF(1) <> 19564 THEN                    '   check for correct file length
  BEEP
  COLOR 3,0
  PRINT "   Not safe to continue - Nonstandard ED program.  Aborting."
  COLOR 1,0
  GOTO Problems
END IF
RETURN

MakeEds:
'  original string = RAW:0/0/639/199/ED 1.14
'  replacement string for lefted = RAW:0/0/319/199/LeftED
'  replacement string for RightEd = RAW:0/320/320/199/RightED

Ed$ = INPUT$(LOF(1),1)
Location = INSTR(Ed$,"RAW:0")
IF Location = 0 THEN              '   check for existence of window string
  BEEP
  COLOR 3,0
  PRINT "   Hmmm, nonstandard ED program here.  Replace with a REAL one!"
  PRINT "   Unwise to continue.  Shutting down."
  COLOR 1,0
  GOTO Problems
END IF

WorkFlag$ = "199"         '  set this to your preferred window height
MID$(Ed$,Location,23) = "RAW:0/0/319/" + WorkFlag$ +"/LEFTside"
                          '   replace filename string with LEFT one
PRINT#2,Ed$;              '   write to a new file
MID$(Ed$,Location,23) = "RAW:320/0/320/" + WorkFlag$ +"/RIGHTside"
                          '   replace filename string with RIGHT one
PRINT#3,Ed$;              '   write to another new file
CLOSE
RETURN

Any:                      '   See if a key is pressed.  ALMOST any key.
                          '   (CAPSLOCK, SHIFT, etc. don't count)
WHILE Any$ = ""
   Any$ = INKEY$
WEND
RETURN

Problems:                 '   Wave bye-bye.  The bug stops HERE!

BEEP
CLOSE
PRINT "   Oops!  All has not gone well in the file handling.  Closing all"
PRINT "   files.  Check to see that you have ED in your SYS:C directory."
PRINT "   That means, the C subdirectory of your main, system disk--the one"
PRINT "   on which your Amiga was booted (generally)."
PRINT "   Also check to see that there is sufficient disk space for all of
PRINT "   the output files.  Check for a proper ED, too.  NOT a hacked one"
PRINT "   from a previous invocation of this, or another, program."

END
```

Listing 2: Paster.BAS

```
'  PASTER.BAS
'  by Jay Gross
'
'  AmigaBASIC program for pasting LeftED- and RightED- created files
'     into one, side-by-side text file
'
ON ERROR GOTO Problems                '  Error handling
'
Hello:
'
COLOR 3,0
PRINT "           PASTER.BAS by Jay Gross"      '  credit the author
COLOR 1,0
PRINT " "
PRINT " "
PRINT "   This program looks in your CURRENT DIRECTORY for files created"
PRINT "   with LeftED and RightED.  These files MUST be named Leftside"
PRINT "   and Rightside, or the program will abort (peacefully)."
PRINT "   If the program finds the appropriate files, it will join them"
PRINT "   together into a single textfile having a name you supply."
PRINT "   The original files will NOT be removed or changed, but using"
PRINT "   LeftED and RightED again will overwrite them."
PRINT " "
COLOR 2,0
PRINT "   Okay, ready to go.  Hit a key when you're ready, too."
COLOR 1,0

GOSUB Any                             '  Check for key being pounded

PRINT " "
PRINT "   Proceeding!  First things first."
PRINT "   What OUTPUT filename do you want created?"

GOSUB Filename

COLOR 2,0
PRINT "   At what column position do you want the columns joined?"
COLOR 1,0
LINE INPUT "   Widest column on left side (40 maximum)?"; Column$
PRINT " "
COLOR 1,0

GOSUB Column

PRINT "   Looking for Left/Right files.  Opening OUTPUT file."

GOSUB Findfile

PRINT "   Okay, the two sides are found, and the main work is proceeding."
PRINT "   This won't take long, and you can watch as it goes."
PRINT "   If you want to stop, interrupt the program with CRTL-C, and"
PRINT "   issue the direct BASIC command, CLOSE."

GOSUB WorkLoop

CLOSE

PRINT "   DONE!  Check it out!  (You can use the TYPE command from CLI)"

END

              '  Now for the GOSUBs

WorkLoop:
WHILE NOT EOF(1)
  IF NOT EOF(2) THEN
    LINE INPUT#1, leftside$               '  get the left string
    LINE INPUT#2, rightside$              '  get the right string
    PRINT LEFT$(leftside$+SPACE$(40),Maxcol); rightside$
    PRINT#3, LEFT$(leftside$+SPACE$(40),Maxcol); rightside$
                                          '  pad with spaces if too short
  ELSE
    LINE INPUT#1, leftside$               '  allow for different filelengths
    PRINT leftside$
```

```
    PRINT#3, leftside$
  END IF
WEND
WHILE NOT EOF(2)
  PRINT SPACE$(Maxcol); rightside$
  PRINT#3 SPACE$(Maxcol); rightside$
WEND
RETURN

Findfile:

OPEN "df0:leftside" FOR INPUT AS #1
OPEN "df0:rightside" FOR INPUT AS #2
OPEN NameIt$ FOR OUTPUT AS #3
'
'   The ON ERROR GOTO statement at the top of the program will stop
'   program execution here if these OPEN statements are unsuccessful.
'
RETURN

Any:

WHILE Any$ = ""
   Any$ = INKEY$
WEND
RETURN

Column:

Maxcol = VAL (Column$)
IF Maxcol > 40 THEN
  Maxcol = 40
END IF
IF Maxcol <= 0 THEN
  Maxcol = 1
END IF
RETURN

Filename:
COLOR 3,0
LINE INPUT "   Include path, if desired (e.g. df1:)"; NameIt$
PRINT " "
COLOR 1,0
PRINT "   Opening " NameIt$ " for OUTPUT file."
PRINT "   This program will OVERWRITE the OUTPUT file if it exists."
PRINT "   If that bothers you, now's your chance to stop."
PRINT "   Okay to go ahead  (Y or N) ?"
WHILE Okay$ <> CHR$(27)
  Okay$ = UCASE$(INKEY$)
  IF Okay$ = "Y" THEN
    RETURN
  END IF
  IF Okay$ = "N" THEN
    COLOR 3,0
    PRINT "   Another chance, then.  What OUTPUT filename?"
    COLOR 1,0
    GOTO Filename
  END IF
  IF Okay$ = CHR$(27) THEN
    GOTO Problems
  END IF
WEND
RETURN

Problems:
BEEP
CLOSE
COLOR 3,0
PRINT "   Oops!  All has not gone well in the file handling.  Closing all"
PRINT "   files.  Check to see that you have the correct root directory."
PRINT "   You might need to change directory (CD) to the one in which your"
PRINT "   left and right ED files are located."
PRINT "   Check to see that there is sufficient disk space for the"
PRINT "   output file.  Also you might have given this program an invalid"
PRINT "   path, or device name.  You ARE using backup files???"
PRINT "
COLOR 1,0
END
```


1 MB/min HD BU rate achieved!

ExpressCopy Only $44.95 US

"The Fast, Safe, and Reliable road for HardDisk Backup and Archival."

ExpressCopy can perform hard disk backups and copy files from hard disk to floppy disk at a rate of up to 1MB per minute, 600k Bytes per minute with verify of backup disks. Backup disks have the same directory structure and file attributes of the original files, and can be used normally. Either Normal or Fast FileSystem compatible disks can be written. Files to be copied can be selected by their DateStamp, filename pattern matching, the Archive Bit, and by source directory. Options for setting the Archive bit, verifying the data written to floppy disk, and estimating the number of disks required for the backup. Up to 4 copies of the backup disks can be created at a time, or disks can be pre-loaded in up to 4 disk drives. New disks are automatically formatted and verified. If a bad disk is found during verify, you can insert a different disk and restart the backup from the beginning of that disk. No need to completely restart the backup. Parameters to be used for backups can be saved in configuration files.

Designed for effective multitasking. Change ExpressCopy's priority while it's running. You can HALT the backup and use the Disk Drives normally, then RESUME the backup.

Backup restoration can be done using any file copy program, your favorite Directory Utility, or by ExpressCopy's Restore program.

ExpressCopy has no copy protection, and can be used from both the CLI and Workbench.

Express-Way Software, Inc.
PO Box 10290, Columbia, MO 65205-4005
(314) 474-2984

Workbench is a trademark of Commodore-Amiga,Inc.

AMIGA Experts

Speak with the people who use the *Amiga*.. Our crew has over 30 years combined experience with the *Amiga*. !
All of the latest Software & Hardware in stock.

We ship via UPS Mon.-Fri.

Commodore authorized repair, low flat rate plus parts.
Warranty service
Store Hours:
Mon.-Thur. 10 - 6
Friday 10 - 8
Saturday 9 - 5

COMMODORE AMIGA

The Memory Location
396 Washington Street
Wellesley, MA 02181
(617) 237-6846
(617) 237-6440

10 % Discount on Software to all club & user group members!

Hard Copy

A new column on books and documentation

by Tim Grantham

Tim Grantham will be known to many readers as the author of the original Amiga Dispatches columns in the 8-bit ***Transactor****, and 'Keep', his widely-used shareware telecommunications utility. Tim has been personally and professionally involved with computers and text for nearly twenty years, most recently as an editor of* ***Design Engineering*** *magazine, and as the author of the manual for Gold Disk's 'Professional Draw'.*

Welcome to the first instalment of a regular column on what might be called 'paperware' - a computer literature column that will mainly consist of book reviews but that will also discuss relevant documents of all kinds. This includes program documentation, readme files, other magazines... anything that purports to elucidate or instruct.

Furthermore, while text objects will be my primary focus, I will occasionally wander off into such areas as user interfaces, television programs, online services - in other words, anything I deem is intended to support and enrich a human's interaction with a computer.

Why? Because I believe we are entering an era when, for better or worse, most of a personal computer's resources will be devoted to the task of *not* being a computer. Hypermedia, graphic interfaces, 'groupware', speech recognition and other information interface techniques are aimed at rendering the computing mechanism invisible to the user, and to extend the user's realm of access and control. As the primitive precursors to the aforementioned tools, books, documentation and online help deserve careful consideration - something that, as anyone who has read an MS-DOS manual or the Aztec C documentation knows, has been sadly lacking in the world of computers.

Documentation: a brief history

Until recent years, software documentation and manuals have been widely acknowledged to be dreadful. The defence, such as it was, was that it was intended for 'expert users'.

The truth is that documentation was not and, in most cases, still is not considered *important* enough to warrant a qualified and capable writer with a budget and time. "Nobody ever sold software with good documentation... It's features, not manuals, that people buy." The first part of this position is debatable, the second is true only in the short term. In this age of unprotected software and the 'non-expert' user, it's quality documentation and the commitment to support that it implies that *keeps* the customers who bought the features.

While there have been significant improvements in the last five years, the move to better documentation did not come from those companies that actually had the resources to provide it. After all, why should IBM, AT&T and DEC go to the bother of documenting their products effectively when they could charge their customers an arm and a leg for service contracts?

It was the advent of the microcomputer, and the mass market it brought to computing, that brought about the first real breakthroughs. Companies that had to reach novice, even technophobic customers, in large numbers to make a profit, began to produce documentation that was not only readable but, in some cases, even enjoyable for its own sake. The wonderful packaging of Infocom's text adventures is the first example that comes to mind. I can also attest from personal experience that the manual for Adobe Illustrator is not only understandable by artists who have never touched a personal computer before, it doesn't look out of place on a coffee table with an Ansel Adams anthology.

Standards for quality documentation

Over the years, I've come to look at three critical qualities as criteria for excellence in documentation and other non-fiction writing. They are of equal weight and, like most criteria, they overlap to some extent.

Clarity. By this I mean proper spelling, correct grammar and straightforward syntax. The last two still require a skilled human writer or editor, but spelling checkers have been broadly available from some time. Yet I still see press releases cross my desk that have clearly been produced on high end desktop publishing systems and that contain spelling errors! They're obvious mistakes, too, like 'microporcessor' or 'dictonary', that even the simplest of spelling checkers would catch. And these press releases are from companies like Apollo Computer and Apple - companies that presumably should know better.

A spelling checker is not enough, however. Any editor worth his or her salt still proofreads again and again; the spelling checker is only an aid.

Another component of clarity is appropriate organization of the material. Computer manuals and textbooks are notoriously poorly organized. Their editors or writers seem to think that readers are machines, not people, and will often make only one reference to a subject, assuming that, once read, it is permanently stored in our brains, complete with all appropriate linkages to related subjects. We have all had the experience of trying to find information on some crucial function or structure, flipping through page after page, only to find it in the place you least expect it.

The Manx Aztec C 3.6a manual is particularly guilty of this. In one section, for example, this manual states that the list of command line switches required by the compiler to generate 68020/68881 code is given in the section describing the operation of the compiler. After wondering why they simply couldn't have indicated the options right there, I turned to the compiler section to find - you guessed it - *not one word* about the required arguments.

I finally found what I was looking for in the readme file. Even worse, I later discovered that that information was also in the 3.4a readme file. In other words, they hadn't bothered to incorporate the readme file from the *previous* version of the compiler when they "significantly upgraded" the manual for 3.6a!

A good index, of course, is an invaluable aid to the reader. However, I'm sure you are all know how common they are.

Precision. This is simple, but hard: the facts must be right - *all* the facts. When unsure just how much explanation a reader requires, a writer should err on the side of generosity. The terminology must be appropriate and accurate.

Readability. This is the most intangible quality of good documentation. It requires clarity and precision as a foundation but depends for its direction on a simple precept that has largely been ignored by producers of computer hardware and software: put yourself in the readers' shoes. An effort should be made to know who the readers are: their educational background, their age, their expectations.

Readability is a very subjective thing. One can discuss the right length of words, sentences and paragraphs to use. One can talk about the right tone to adopt: use humour where possible, avoid pedantry, shun puffery. In truth, however, the essential requirements reside within the writer: talent, experience and a genuine desire to communicate effectively.

Readability is also where graphic design comes into play. I'm not a designer, so I won't attempt to go into depth about this. But even such apparently simple things as the choice of typeface, and type styles and sizes, can contribute greatly to the rapid comprehension of text.

Now you know where I stand. I'm not a perfect writer, either, of course; I expect to spend my whole life learning how to write. And with that out of the way, I'll start this book and documentation review column with a look at two fine examples of clarity, precision and readability...

Understanding PostScript Programming, Second Edition
by David A. Holzgang
Softcover, 472 pages, $24.95 US
Published in 1988 by SYBEX Inc.
2021 Challenger Drive #100, Almeda, California 94501

Several months ago, I found myself in need of a book that would provide a solid introduction to PostScript. I found the prospect of purchasing and reading all three official Adobe Systems reference guides daunting, so I cast my eye around for another, single-volume, candidate.

Understanding PostScript Programming is not only a fine introduction to PostScript, it is an exemplary computer book in almost every respect. It is one of the few books I have encountered - and certainly the only computer book - in which I was unable to find a single spelling or syntactical error. I was also unable to detect any programming errors or any inconsistencies in Holzgang's explanations.

These accomplishments alone are often sufficient to make for painless reading. Add to this the author's thoughtful and humane organization of the material and his unforced and direct style of writing, and the result is that rare book that is both thorough and a pleasure to read.

Holzgang is careful to emphasize that the book is not meant to replace the official Adobe texts, but to complement them. He does not attempt to provide a definitive reference. Rather, he offers a practical perspective on PostScript for those new to the language. To quote from his introduction (pg. XXII): "Because this book is directed toward understanding the basics of the language, it will concentrate its examples and discussions on the most useful 15 percent or 20 percent of [PostScript] operators. This is a reasonable and effective method for mastering a new programming language."

The author takes a spiral approach, providing elementary examples of text handling, graphics and fonts in the first chapter that are then deepened in subsequent chapters. Every chapter begins with a summary of what will be discussed and concludes with a review of the operators introduced.

Holzgang assumes that the reader has access to a PostScript printer and a personal computer. (I did not have access to the printer during my first reading of this book, yet I still found it quite easy to follow.) He begins by working in PostScript's interactive mode, providing short programs to introduce the essential commands for placing text in various places on the page. Throughout the book, the PostScript source is shown first, followed by reduced reproductions of the actual output.

Holzgang's years of teaching are clearly in evidence as he walks the reader through the programs, explaining what is happening, anticipating points of confusion, reminding the reader of past lessons, yet never patronizing or babying.

The book goes on to introduce the concept of stacks and stack operations, the conceptual foundation of PostScript. Then dictionaries, the mechanism by which constants, variables and procedures are defined, are introduced.

This section also introduces the reader to page analysis. The PostScript programmer must have a thorough understanding of the type of output desired before commencing programming, and this chapter helps the reader grasp the PostScript view of the page.

The chapter concludes with a discussion of PostScript programming style and conventions. In PostScript, comments have much greater importance than they do in most other languages. Adobe Systems has actually published standards for comments, for they are the handle that other programs use to process PostScript programs. The author is careful to observe the Adobe conventions in every respect.

As the reader proceeds through the example programs, the author provides procedures that are used in subsequent programs. This not only demonstrates the importance of structured programming, it provides the basis for a library of useful procedures that the reader can draw on in his or her own work.

PostScript font mechanisms and graphics commands are introduced jointly in the following chapter. There is a conceptual justification for this, for fonts and text are as much graphic objects in PostScript as lines and arcs.

The last part of this chapter consists of an introduction to the coordinate structure of PostScript and the all-important translation, rotation and scaling manipulations. As the author makes clear, these concepts, along with the saving and restoring of the graphics state, are essential to the creation of independent and flexible procedures. He demonstrates this with the creation of a simple stylized logo built with the basic line and arc operators.

The concepts covered so far are now applied to the task of creating an invoice for a fictitious mountain sports company, complete with the logo from the previous chapter. The particulars are filled in with data created from a spreadsheet, just as they might be in real life.

This exercise has tremendous potential for boredom, but Holzgang's clear analysis of the construction of the invoice, and the simplicity of his coding style, serve to demonstrate the power and elegance of the language. By the end of the chapter, I was eager to begin programming simple documents myself.

The title of the next chapter, "Creating and Modifying Fonts", is a bit misleading: it's not a discussion of the aesthetics of typeface design. Rather, it's an exploration of such issues as the downloading of additional fonts, the extension of the character set to include characters found in foreign languages, and the creation of outlined letters and fonts. The author takes this opportunity to delve more deeply into font structure and mechanics, with discussion of font encoding, font metrics and the font cache. He demonstrates the principles with the creation of a dinner menu for a French restaurant.

The final two chapters deal with advanced text and graphics techniques. These include the use of string handling operators to improve the process of justification of text, including spacing adjustment and kerning; a brief introduction to PostScript files and file handling; and a general discussion of the modification of PostScript files generated by application programs such as AutoCAD or Adobe Illustrator. The advanced graphics techniques discussed include introductions to the concepts of clipping, line joins and bitmapped images.

Several useful appendices are provided: a review of all the operators used in the book; a discussion of the new 2.0 commenting conventions recently released by Adobe; a discussion of the Encapsulated PostScript format, which is used to exchange PostScript graphics files between application programs (Gold Disk's Professional Draw, for example, uses EPSF); a chapter on configuring an IBM PC (or compatible) or a Macintosh to a laser printer for use with the book's example programs; and finally, a bibliography that includes references to books on desktop publishing and type and graphic design, as well as other PostScript texts.

The book concludes with a reasonably good index.

And I will conclude my review with a quote from Holzgang's discussion of typeface design (p. 145): "Aesthetics and utility are merged in perfect harmony in the best type designs. The beauty of a typeface is directly related to how well it performs its utilitarian functions of clarity, emphasis and legibility." The same thing can be said about books and book design. *Understanding PostScript Programming* is a fine example of both.

COMPUTE!'s Amiga Machine Language Programming Guide
by Daniel Wolf and Douglas Leavitt, Jr.
Softcover, 414 pages, $19.95 US.
COMPUTE! Publications, Inc., 1988
P.O. Box 5406, Greensboro, NC 27403 USA.
A companion disk containing a Metacomco-compatible assembler and source code for the programs in the book is also available from COMPUTE! Publications at $19.95 US.

From a microcomputer programmer's point of view, the Amiga operating system is unique. There are multitasking, multi-user operating systems on other personal computers, but none offer the combination of features found on the Amiga. OS/9 on the Tandy Color Computer and the CBM SuperPET is a true multitasking OS that has been around for years, but it doesn't have a standard graphic interface. The same applies to

the versions of UNIX and Concurrent DOS that run on the PC clones. The much-ballyhooed OS/2 on the 80286 machines has only just added the graphic Presentation Manager, but OS/2 only supports a single user. The Macintosh Multifinder OS does not use true multitasking, nor does it support multiple users.

Those new Amiga programmers coming from an engineering workstation environment, such as Sun or Apollo, quickly adapt because these machines are closest in spirit to the Amiga. Indeed, the Amiga's OS was mostly developed on Sun workstations.

Those programmers coming up to an Amiga from single-tasking, single-user microcomputers however, face a very steep learning curve. A very different way of thinking is required when faced with a machine that, from a user programmer's point of view, has only one fixed address. Add to that the need to co-exist with other programs and to adapt to a mouse-driven, windowing user interface, and the assembly language programmer can hardly be blamed for feeling overwhelmed.

COMPUTE!'s Amiga Machine Language Programming Guide brings welcome relief to the confused. To my knowledge, it is one of the few books to provide a decent conceptual overview of the Amiga programming environment, and certainly the only one to do so for the assembly language programmer. Even expert programmers would do well to consider this book as a primary reference, if they are new to the Amiga.

While the authors state in the introduction that the book is designed for both new and experienced programmers, I believe that at least some familiarity with machine language concepts is essential. A novice would probably enjoy this book if he or she has a companion machine language reference, such as the *68000/68010/68020 Primer* by Stan Kelly-Bootle and Bob Fowler, published by The Waite Group. Those who have programmed in assembly language on 8-bit processors will have little difficulty with the chapters on 680x0 machine language.

The book deals only with user programming in the Amiga environment. As such, it does not provide any coverage of the 680x0 supervisor mode.

It begins with a general introduction to the architecture of the 680x0 series of microprocessors. The heart of the section lies in the chapter titled "Most Frequently Used MC 68000 Instructions", which the authors detail with dispatch. I was pleased to see that the authors go to the trouble of explaining how to maintain code compatibility with the other processors in the 680x0 series, ruling out, for example, the use of the status register read and write instructions and the use of the upper 8 bits in pointers for purposes other than the addressing of memory. 68010-equipped Amigas are quite common now, and more 68020-based Amigas are appearing.

The second section, "Amiga Machine Language Concepts", forms the crux of the book, providing the machine language programmer with the minimum knowledge of AmigaDOS and Amiga assemblers required to begin his or her explorations. Assembling and linking are introduced with a "hello, world" type program, followed by chapters on such essential assembler directives as EQU (equate), INCLUDE and MACRO; declaration of storage and constants; and conditional assembly.

The section concludes with a comparison between the Commodore-blessed Metacomco assembler, and the COMPUTE! assembler, ASM68010, available on a companion disk.

ASM68010, written by Daniel Wolf, is a Metacomco-compatible assembler that adds support of the 68010 microprocessor. It also features a so-called 'autolinking' capability: provided no references are made in the source code to external symbols or values, it can produce an executable program directly. It is also compatible with the Alink or Blink linkers, should external libraries be referenced.

The authors have provided well-commented include files that contain the most commonly needed library and structure offsets. They have also provided STARTUP.ASM, a kind of standard module that prepares the Amiga environment for a user's program and cleans up after it. As a result, none of the example programs in the book need to use the Amiga libraries, so none of them require linking if ASM68010 is used.

In my view, these files, and the wealth of useful macros contained therein, provide a very congenial, self-contained environment for the beginning Amiga assembly language programmer. When using the ASM68010 assembler, the crucial write/run/debug cycle is faster, less memory is consumed, and the official Commodore include files are not required. The only fly in the ointment, and it's a small one, is that the authors have chosen to use their own names for system structures and functions, rather than the ones found in the official include files and manuals. Perhaps this was done to avoid copyright problems, but I think it may lead to confusion.

Section 3, "Amiga Special Programming Techniques" introduces the Amiga run-time libraries, or 'movable memory maps' as Jim Butterfield calls them, and the Amiga system structures. These techniques are 'special' only in the sense that they pertain to the Amiga OS; in every other sense, they are the basic tools of Amiga programming. This section also includes a chapter on Amiga program startup code, specifically the STARTUP.ASM module mentioned previously.

At eleven pages, Section 4, "Programming With AmigaDOS", provides a skimpy introduction to the Amiga disk operating system. There is only the barest mention of BCPL, and the only DOS functions described are Input, Output, Open, Read and Write (and only for console I/O) and Execute, to call other programs from within a user program. This is the weakest section of the book.

On the other hand, Section 5, "Intuition", provides a thorough grounding in the basics of the Amiga's graphic interface, with

ninety pages of discussion of windows, menus, gadgets, requesters, Intuition message ports and IntuiText. The numerous example programs not only illustrate the programming principles but also contain useful subroutines and macros that ease the burden of programming in the Intuition environment.

The next section provides an introduction to graphics, using some basic routines from the graphics library. These include drawing-colour control functions, line drawing functions, point drawing functions, region filling functions and colour register control functions. Also covered, for the first time in the book, is the creation of custom screens.

"Amiga Floating-Point Math" is another skimpy section, this time because it assumes knowledge of assembly language math. It does introduce use of the Amiga floating-point and transcendental math libraries, but not the *mathieeedoubbas* library for IEEE 64-bit floating point math. Some useful macros and subroutines are listed, including a subroutine that converts an ASCII string representing a floating point number to its 32-bit Motorola Fast Floating Point representation.

COMPUTE!'s Amiga Machine Language Programming Guide concludes with a section documenting and listing several fairly impressive demonstration programs. POLYFRAC.ASM demonstrates fractal line drawing, while LENS.ASM produces a window that shows a magnified version of the area of the screen around the pointer. QUADRIX.ASM is a 3D graphics creation program that generates quadric surfaces such as paraboloids and cones. Finally, ASMINT.ASM provides an Intuition graphic interface to the ASM68010 assembler.

The appendices consist of a listing and description of the 68000/010 instruction set and assembler directives; a list with explanations of common assembly-time errors; even better, a listing with explanations of guru meditation error codes; documentation for the ASM68010 assembler on the companion disk; and an ASCII table. The book is rounded out with a glossary of terms encountered in Amiga assembly language programming and a mediocre index.

Some quibbles: the source code listings are capitalized, which I find hard on the eyes; no mention is made of ROMWack or any debugging techniques; and even apart from the serious lack of AmigaDOS coverage, the book is generally terse - not surprising perhaps, given the enormous amount of territory that has to be covered. Let's hope Jim Butterfield will write another assembly language book, this time for the Amiga.

Some nice details: I have always liked COMPUTE! book design - the use of ragged right text, the typeface of the body copy (but not the heads), the binding rings that let the book lie flat. The overall organization of the material is good, the tone is right and I found only two spelling errors. The glossary contains nice encapsulations of important concepts and terms that I think tyros will find helpful.

Highly recommended.

MCP ASSOCIATES INC.
New York's Premier AMIGA Dealer
Competitive Pricing
ABSOLUTELY EVERYTHING FOR AMIGA
Systems . . . Hardware . . . Software . . . Repairs . . .
Disks . . . Special Offers . . . Distributors . . .
Amiga Specialists . . .
Call 1-718-956-9000
for catalogue or write to:
28-21 ASTORIA BLVD., ASTORIA, N.Y. 11102-1933
FAX: 1-718-956-9028

PowerMaster Amiga Home Control
Programmed Remote Control
Of Lights & Appliances!
• Adds Convenience, Increases Security
• Uses X-10 Power-House System
• Amiga Not Required After Programming
• PowerMaster & Amiga Interface:
$65 U.S. + $3 s/h ($6.50 Cdn.)
Compplications
1727 Parkview
Redlands, CA • 92374
(714) 794-5311

INTRODUCING.....

An Evolution in Disk Utilities for Amiga™ Personal Computers!
FEATURING
• An easy to use, friendly and intuitive user interface.
• A powerful and fast disk backup tool that lets you make backups of your copy-protected Amiga software.
• A disk editing tool that lets you edit raw MFM tracks, AmigaDOS sectors and AmigaDOS files (automatically calculating new checksums).
• A disk cataloging tool that lets you maintain lists of your personal, public domain and commercial software.
• A unique backup tool for duplicating other disk formats including MS-DOS/PC-DOS and Atari ST.
• An easy to read, informative user manual is included.
• This product is not copy-protected in any way.
NOW SHIPPING!
$49.95 EA.
Includes shipping and handling!
Arizona residents add 6.5% sales tax.
TO ORDER
Send check or money order to:
Fuller Computer Systems, Inc.
P.O. Box 9222
Mesa, Arizona 85214-0430
Or CALL (602) 835-5018
Amiga is a trademark of Commodore-Amiga, Inc.
Dealer Inquiries Invited

Reader Service Card

Fill out this card (please print) or attach your peel-off mailing label, and circle the reader service number for the ads that you'd like more information about (see Advertiser Index below).

Name _____

Company _____

Address _____

_____

Postal/Zip Code _____

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

TA V1I6

Advertiser	Page	Reader Service #
A.S.D.G.	5	1
AmiWare	41	2
Briwall	36-37	3
Comp-U-Save	59	4
Compplications	80	5
Creative Computers Inc.	10-11	6
D-Five	35	7
Discovery Software	Outside Back Cover	8
Electronics 2001	35	9
Emerald Intelligence	Inside Front Cover	10
Express-Way	75	11
Fisher's	69	12
Fuller Computer Systems	80	13
Great Valley Products	Inside Back Cover	14
Intercomputing Inc.	51	15
Interface Technologies	26	16
Lake Forest Logic Inc.	15	17
M.C.P. Associates Inc.	80	18
Microbotics Inc.	27	19
Micromaster	35	20
Microsmiths Inc.	9	21
NewTek	15	22
O.T.G. Software	31	23
Palomax Inc.	41	24
PC Plus	69	25
Prolific Inc.	31	26
Redmond Cable	58	27
Rick Stiles (UEdit)	69	28
Software Excitement	1	29
SPOC	58	30
The Memory Location	75	31
William S. Hawes	50	32

for the AMIGA Transactor

The Magazine for Amiga Programmers

Please print your name and address below (or transfer your mailing label)
Check here for change of address ☐

Name: _____

Address: _____

City: _____ *Province/State:* _____

Zip/Postal: _____ *Phone:* _____ *CompuServe ID:* _____

NOTE: Prepayment required. Purchase orders will be accepted ONLY if accompanied by payment.

Subscribers only: please enter subscription number from mailing label ☐

Subscription Order Section

PLEASE NOTE THAT TRANSACTOR FOR THE AMIGA IS NOW PUBLISHED MONTHLY

	Canada ($Cdn.)			U.S.A. ($US)			Foreign ($US)			Air Mail ($US, overseas only)		
	Mag	Disk*	Both*	Mag	Disk	Both	Mag	Disk	Both	Mag	Disk	Both
12 issues	☐35.00	☐123.00	☐158.00	☐27.50	☐101.00	☐128.50	☐38.50	☐119.50	☐157.50	☐ 73.00	☐ 145.00	☐ 218.00
24 issues	☐65.00	☐220.00	☐270.00	☐49.00	☐160.00	☐190.00	☐69.00	☐220.00	☐290.00	☐ 135.00	☐ 250.00	☐ 390.00

*** Ontario residents please add 8% of disk subscription price for provincial sales tax. No PST on magazine subscriptions.**

☐ **Please send Microfiche instead of magazines** ☐ **Send me info on Transactor Classic (C64/128)**

Combo Subscription Bonus!

If you are subscribing to BOTH the magazine and the disk, check one of the following FREE gifts:

☐ **Transactor Classic Amiga Disk #1** ☐ **Fish Catalog Disk** ☐ **Extended Subscription (13 Issues)**

Product Order Section

Ontario residents please add 8% PST on all items

$Cdn.	$U.S.	
☐ 14.95	12.95	Transactor Classic Amiga Disk #1
☐ 14.95	12.95	Transactor Classic Amiga Disk #2

Fish Disks $6.95 US, $7.95 Cdn. each

Note: Disk numbers not yet released may be ordered in advance - simply add numbers to end of list. Disks 57, 80 and 88 are not available.

☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7	☐ 8	☐ 9	☐ 10	☐ 11	☐ 12	☐ 13	☐ 14
☐ 15	☐ 16	☐ 17	☐ 18	☐ 19	☐ 20	☐ 21	☐ 22	☐ 23	☐ 24	☐ 25	☐ 26	☐ 27	☐ 28
☐ 29	☐ 30	☐ 31	☐ 32	☐ 33	☐ 34	☐ 35	☐ 36	☐ 37	☐ 38	☐ 39	☐ 40	☐ 41	☐ 42
☐ 43	☐ 44	☐ 45	☐ 46	☐ 47	☐ 48	☐ 49	☐ 50	☐ 51	☐ 52	☐ 53	☐ 54	☐ 55	☐ 56
☐ 58	☐ 59	☐ 60	☐ 61	☐ 62	☐ 63	☐ 64	☐ 65	☐ 66	☐ 67	☐ 68	☐ 69	☐ 70	☐ 71
☐ 72	☐ 73	☐ 74	☐ 75	☐ 76	☐ 77	☐ 78	☐ 79	☐ 81	☐ 82	☐ 83	☐ 84	☐ 85	☐ 86
☐ 87	☐ 89	☐ 90	☐ 91	☐ 92	☐ 93	☐ 94	☐ 95	☐ 96	☐ 97	☐ 98	☐ 99	☐ 100	☐ 101
☐ 102	☐ 103	☐ 104	☐ 105	☐ 106	☐ 107	☐ 108	☐ 109	☐ 110	☐ 111	☐ 112	☐ 113	☐ 114	☐ 115
☐ 116	☐ 117	☐ 118	☐ 119	☐ 120	☐ 121	☐ 122	☐ 123	☐ 124	☐ 125	☐ 126	☐ 127	☐ 128	☐ 129
☐ 130	☐ 131	☐ 133	☐ 134	☐ 135	☐ 136	☐ 137	☐ 138	☐ 139	☐ 140	☐ 141	☐ 142	☐ 143	☐ 144
☐ 145	☐ 146	☐ 147	☐ 148	☐ 149	☐ 150	☐ 151	☐ 152	☐ 153	☐ 154	☐ 155	☐ 156	☐ 157	☐ 158
☐ 159	☐ 160	☐ 161	☐ 162	☐ 163	☐ 164	☐ 165	☐ 166	☐ 167	☐ 168	☐ 169	☐ 170	☐ 171	☐ 172

The Transactor for the *AMIGA* Disk $12.95 US, $14.95 Cdn. each
(these disks correspond to issues of **Transactor for the *AMIGA***)

☐ Vol. 1, Iss. 1 ☐ Vol. 1, Iss. 2 ☐ Vol. 1, Iss. 3 ☐ Vol. 1, Iss. 4 ☐ Vol. 1, Iss. 5 ☐ Vol. 1, Iss. 6

Add flat $2.00 shipping & handling for any number of items ordered from above box.

Transactor for the *AMIGA* Back Issues and Microfiche: $4.50 U.S., $4.50 Cdn. (MF = Microfiche)

☐V1#1 ☐MF ☐V1#2 ☐MF ☐V1#3 ☐MF ☐V1#4 ☐MF ☐V1#5 ☐MF ☐V1#6 ☐MF

Ontario residents please add 8% provincial sales tax for any items ordered from this box.

☐ Visa ☐ MasterCard ☐ Cheque/MO enclosed Cheque #_____ Dated ___/___/___ Amount _____

Acct. # _____ Expires ___/___ Signature _____

Please allow 6-8 weeks for delivery in the U.S., 4-6 weeks in Canada.

FOLD ALONG SCORES WITH CORRECT REPLY CARD ON OUTSIDE. USE TAPE, PLEASE - NO STAPLES OR GLUE.

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 390 BUFFALO, NY
POSTAGE WILL BE PAID BY ADDRESSEE

The Transactor
P.O. Box 338 Station 'C'
Buffalo, NY, 14209-9990

place stamp here

Transactor
10-85 West Wilmot Street
Richmond Hill, Ontario, Canada
L4B 1K7

Business Reply Mail
No Postage Stamp Necessary
If mailed in Canada

Postage will be paid by:

Canada Post Postes Canada 46487

The Transactor
501 Alden Road
P.O. Box 3250
Markham Industrial Park
Markham, Ontario
L3R 9Z9

Use our convenient...

TOLL-FREE ORDER LINE
1-800-248-2719 Extension 911

Here's a convenient way to order magazine or disk subscriptions and other products. Nothing to fill out or drop in the mail! Just call the toll-free number above and have your credit card number ready. Available in the United States only.

NOTE: The toll-free number can be used for ordering only; for subscription queries, etc., call or write to our office directly.

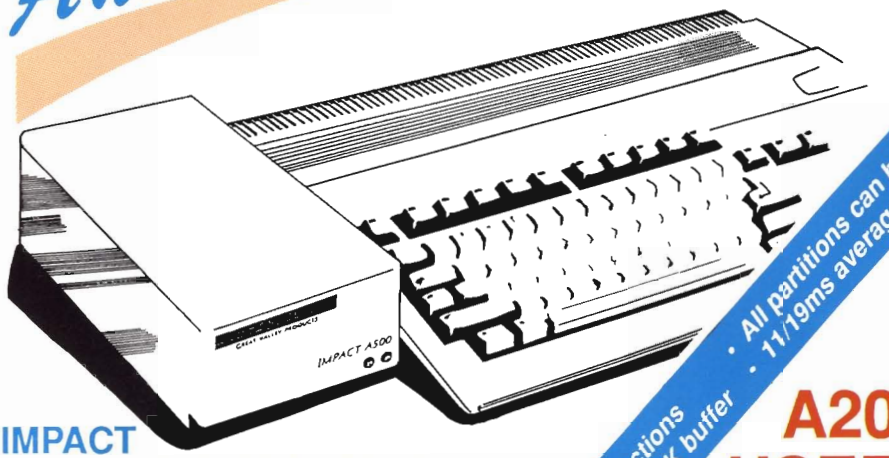
Advertising in Transactor for the Amiga

Interested in advertising in *TransAmi?* Call Max Jacobson or Julie Cale at 416/764-5274 NOW!

AMIGA 500 USERS!
NOW GET A2000 POWER!!

Announcing

THE WORLD'S FIRST AUTOBOOTING HARD DISK FOR THE AMIGA 500



• All partitions can be FFS with V1.3 AUTOBOOT
• 11/19ms average access time on 40Q & 80Q models

• Internal & external SCSI connections
• DMA data transfer to onboard 16K buffer

• AUTOBOOT directly from hard disk
• Connect up to 6 SCSI peripherals

IMPACT A500 – HD/RAM SUBSYSTEM

offers everything any A500 owner has ever dreamed of – in one compact, easy-to-install, add-on subsystem; provides the same high performance as our top-of-the-line A2000 controller

IMPACT™ A500 – HD/20
A500 – HD/30
A500 – HD/40Q
A500 – HD/45
A500 – HD/80Q

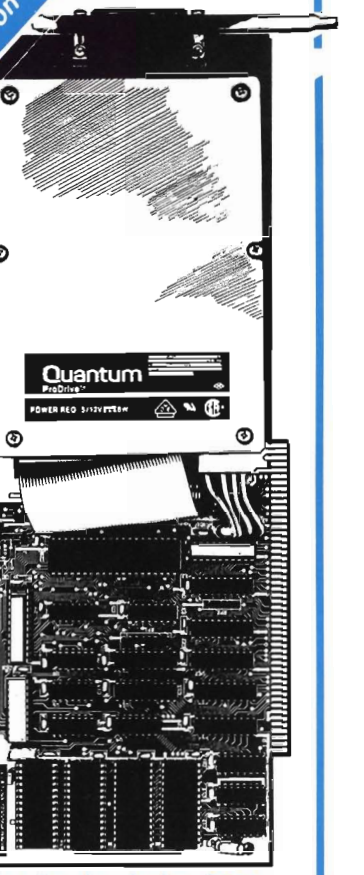
Available with a built-in 20, 30, 40, 45 or 80 Mega Byte Hard Drive and an optional 2MB plug-in FAST RAM/AUTOBOOT expansion module.

A2000 USERS!

Still

THE MOST POWERFUL HARD DISK SOLUTION FOR YOUR A2000!

IMPACT™ A2000 - HC/30
A2000 - HC/40Q
A2000 - HC/45
A2000 - HC/80Q
A2000 - HC/100



The GVP IMPACT-HC "Harddisk-on-a-Card" series for the Amiga 2000 matched with the 3.5" Quantum ProDrive hard disk drives, provide turbo performance and unbeatable value at list prices as low as $18.00 per Maga Byte! The Quantum ProDrives provide average access times as low as 11ms with their built-in 64KB read-ahead cache and offer a MTBF of 50,000 hours which is almost twice the industry average.

Now Shipping **AUTOBOOT EPROMS**

GVP

GREAT VALLEY PRODUCTS INC.

225 PLANK AVENUE, PAOLI, PA 19301 TEL (215) 889-9411 FAX (215) 889-9416

SWORD OF SODAN™

Grasp the Sword of Sodan and feel the true power of arcade action!

Creaking, groaning, shrieking, teeth clenching arcade realism. Deep dark forests. Misty, eerie graveyards. Treacherous, guard-filled streets. Dare to enter the evil castle, Craggamoore. Wield your sword through its winding corridors and rank rotting dungeons. Your story will be the greatest ever told — Your reward will be the greatest ever known!

FEATURES:

- ***Battle with massive animated characters that stand 2/3rds the size of the screen.***
- ***Fight as Hero or Heroine through 11 incredibly challenging levels!***
- ***Study your past performance in Replay Mode! (extra memory required)***

Packed into 3 disks with more than 4 MEGABYTES of detailed graphics and digitized sounds!

Available for Amiga™, Coming soon for IBM® and compatibles, Apple II GS.

AMIGA screens shown.

DISCOVERY SOFTWARE

Fight your way to your local software retailers. Or, call 1-800-342-6442 for VISA and MasterCard Orders. For check or money orders, send $49.95 plus $3 shipping and handling to DISCOVERY SOFTWARE INTERNATIONAL INC. 163 Conduit Street, Annapolis, MD 21401. Allow 4-6 weeks for delivery. Sword of Sodan is a trademark of Discovery Software International, Inc. Amiga is a trademark of Commodore Amiga, Inc. IBM is a registered trademark of International Business Machines, Inc.

Dorian 1988 ©